Titre: Title:	Méthodes hybrides basées sur la génération de colonnes pour des problèmes de tournées de véhicules avec fenêtres de temps
Auteur: Author:	Eric Prescott-Gagnon
Date:	2011
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Prescott-Gagnon, E. (2011). Méthodes hybrides basées sur la génération de colonnes pour des problèmes de tournées de véhicules avec fenêtres de temps [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/507/

UNIVERSITÉ DE MONTRÉAL

MÉTHODES HYBRIDES BASÉES SUR LA GÉNÉRATION DE COLONNES POUR DES PROBLÈMES DE TOURNÉES DE VÉHICULES AVEC FENÊTRES DE TEMPS

ERIC PRESCOTT-GAGNON
DÉPARTEMENT DE MATHÉMATIQUES ET GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES DE L'INGÉNIEUR)
FÉVRIER 2011

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

MÉTHODES HYBRIDES BASÉES SUR LA GÉNÉRATION DE COLONNES POUR DES PROBLÈMES DE TOURNÉES DE VÉHICULES AVEC FENÊTRES DE TEMPS

présentée par : PRESCOTT-GAGNON, Eric
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

M. GENDREAU, Michel, Ph.D., président.
M. ROUSSEAU, Louis-Martin, Ph.D., membre et directeur de recherche.
M. DESAULNIERS, Guy, Ph.D., membre et codirecteur de recherche.
M. LAPORTE, Gilbert, Ph.D., membre.
Mme. SPERANZA, Maria Grazia, Ph.D., membre externe.

À mes parents, merci pour tout.

REMERCIEMENTS

Je tiens à remercier mes directeurs de recherche, M. Louis-Martin Rousseau et M. Guy Desaulniers, pour leurs conseils et leur encadrement tout au long de ce long processus. La qualité de ce travail leur est due en grande partie.

Je tiens aussi à remercier Michael Drexl pour son accueil à l'institut Fraunhofer à Nuremberg et pour l'ambiance de travail extrêmement conviviale.

Je remercie aussi mes parents sans qui je n'existerais pas et grâce à qui je suis devenu la personne que je suis aujourd'hui.

Je remercie finalement M. Michel Gendreau d'avoir eu l'amabilité de présider le jury de cette thèse. Je remercie aussi Mme. Grazia Speranza ainsi que M. Gilbert Laporte pour avoir bien voulu en faire partie.

RÉSUMÉ

Un problème de tournées de véhicules avec fenêtres de temps consiste à faire la livraison de marchandise à un ensemble de clients avec une flotte de véhicules ayant un ou plusieurs points de départ appelés dépôts. Chaque client doit être desservi à l'intérieur d'une période prédéfinie, appelée fenêtre de temps. En pratique, on doit pouvoir respecter un grand nombre de contraintes et de caractéristiques complexes telles que des flottes hétérogènes de véhicules, des restrictions sur les routes, etc., en plus de devoir prendre en compte un grand nombre de clients. Il est donc primordial pour les distributeurs d'avoir accès à des outils performants d'optimisation capables de gérer un grand ensemble de contraintes de façon efficace.

Dans cette thèse, nous présentons une méthode heuristique pour résoudre un ensemble de problèmes de tournées de véhicules de grande taille avec fenêtres de temps de façon efficace. Les problèmes abordés sont riches dans le sens où ils contiennent des caractéristiques non conventionnelles complexes s'apparentant à des problématiques réelles. La méthode proposée est un hybride entre une méthode métaheuristique de recherche à grands voisinages et une méthode exacte de génération de colonnes, la plus performante à ce jour pour résoudre de façon exacte des problèmes de tournées de véhicules assez contraints.

La recherche à grands voisinages est une méthode où l'on vient itérativement détruire (phase de destruction) et reconstruire (reconstruction) des parties d'une solution courante afin d'obtenir de meilleures solutions. Les voisinages, définis dans la phase de destruction, sont explorés dans la phase de reconstruction. Dans notre méthode, les voisinages sont explorés par génération de colonnes gérée de façon heuristique. Une méthode de génération de colonnes sert à résoudre la relaxation linéaire d'un programme linéaire. Elle résout itérativement un problème maître, qui est le programme linéaire restreint à un sous-ensemble de variables, et un ou plusieurs sous-problèmes qui servent à rajouter des variables de coût réduit négatif au problème maître. La résolution se termine lorsque les sous-problèmes ne trouvent plus de variables de coût réduit négatif. Cette méthode est imbriquée dans un algorithme de séparation et évaluation pour obtenir des solutions entières.

Plusieurs opérateurs sont définis pour sélectionner des éléments qui seront retirés de la solution courante dans la phase de destruction. À chaque itération, un opérateur est choisi aléatoirement en favorisant ceux qui ont permis d'améliorer la solution courante dans les itérations précédentes. La génération de colonnes sert ensuite à explorer le voisinage ainsi défini (reconstruction). Plusieurs aspects de la génération de colonnes sont gérés de façon heuristique afin d'obtenir de bonnes solutions en des temps raisonnables aux dépens de la certitude de trouver une solution optimale. Les sous-problèmes sont résolus par une méthode

de recherche tabou et la génération de colonnes est stoppée après une trop faible amélioration de la valeur de la solution courante de la relaxation linéaire au cours des dernières itérations. Afin d'obtenir des solutions entières, un branchement agressif sur la variable ayant la valeur fractionnaire la plus grande est effectué. Sa valeur est fixée à 1 sans possibilité de retour en arrière.

Cette thèse comporte trois chapitres principaux, chacun correspondant à un article publié ou soumis pour publication. Chacun de ces chapitres présente la méthode hybride générique expliquée ci-haut adaptée à un problème spécifique de tournées de véhicules.

La méthode est d'abord développée au chapitre 4 pour le problème de tournées de véhicules avec fenêtres de temps. L'objectif est d'abord de réduire le nombre de véhicules et ensuite la distance totale parcourue. Pour cette raison, la méthode est adaptée en deux phases, une pour chacun des objectifs. Cette méthode a réussi, au moment de l'expérimentation, à améliorer la meilleure solution connue de 106 sur 356 instances de tailles allant de 100 à 1000 clients.

Afin de mettre à l'épreuve la méthode sur un problème plus complexe ayant une portée réelle, les règles européennes devant être respectées par les horaires des chauffeurs sont ajoutées à la problématique au chapitre 5. En effet, en avril 2007, une mise à jour des règles régissant les heures de travail des chauffeurs est entrée en vigueur au sein de l'union européenne. Malgré un aspect concret et très pratique, peu d'articles scientifiques ont été publiés sur le sujet d'un point de vue d'optimisation. Afin de vérifier si une route est réalisable, un algorithme d'étiquetage est proposé. Les règles y sont définies à l'aide de fonctions de prolongation de ressources complexes pouvant générer plusieurs étiquettes à partir d'une seule. On vérifie la validité de chaque nouvelle route engendrée par chaque insertion évaluée dans la méthode tabou résolvant le sous-problème. On démontre aussi comment considérer plusieurs insertions à la fois en bornant les ressources à chaque nœud de la route courante de la méthode tabou. Ces bornes sont aussi utilisées pour restreindre le domaine de l'algorithme d'étiquetage. La méthode proposée pour vérifier la validité des routes peut être transférée à n'importe quel algorithme utilisant des mouvements d'insertion. Comparée à deux autres méthodes sur des instances académiques, la nôtre est clairement supérieure.

La méthode est finalement généralisée au chapitre 6 à un problème réel se posant dans l'industrie de la distribution d'huile de chauffage. Beaucoup plus complexe, le problème nécessite plusieurs sous-problèmes pour la méthode de génération de colonnes. De plus, il y a une flotte hétérogène, des heures de début et de fin de quarts de travail, plusieurs dépôts, des ravitaillements intra-routes ainsi que des clients optionnels. Étant donné la similitude des sous-problèmes, il est possible d'ajouter un type de mouvement dans la méthode tabou résolvant les sous-problèmes pour passer d'un sous-problème à l'autre sans ralentir grandement

la méthode. Une méthode tabou concurrente est aussi présentée pour résoudre le problème en entier. Cette méthode est aussi insérée à l'intérieur de l'algorithme de recherche à grands voisinages pour explorer les voisinages au lieu de la génération de colonnes heuristique. Les résultats numériques démontrent la validité d'une recherche à grands voisinages comme mécanisme de guidage pour contrôler une méthode métaheuristique (en l'occurrence la méthode tabou). Par contre, la reconstruction par génération de colonnes est plus performante pour les instances résolues.

La méthode de génération de colonnes heuristique imbriquée à l'intérieur d'une recherche à grands voisinages est donc une méthode hybride très compétitive qui peut être appliquée à un large éventail de problèmes de tournées de véhicules. Elle pourrait aussi être facilement adaptée à d'autres types de problèmes où une formulation de génération de colonnes est applicable. Bien que plus lente que d'autres méthodes récentes, il est possible d'obtenir de meilleures solutions en agrandissant la taille des voisinages de la recherche à grands voisinages.

ABSTRACT

Given a fleet of vehicles assigned to one or more depots, a vehicle routing problem with time windows consists of determining a set of feasible vehicle routes to deliver goods to a set of scattered customers. Every customer must be visited within a prescribed time interval, called a time window. In practice, vehicle routing problems can have many different types of constraints and complex characteristics such as a heterogeneous fleet, restrictions on the routes, etc., while having to serve a large number of customers. Therefore, it is essential for distributors to rely on competitive optimizing tools able to tackle a large number of constraints efficiently.

In this thesis, we present an efficient heuristic method for solving a number of large-scale vehicle routing problems with time windows. The problems tackled are rich in the sense that they contain many non-conventional complex characteristics arising in real applications. We propose a hybrid between a large neighborhood search metaheuristic and a column generation exact method, hitherto the most efficient to solve constrained vehicle routing problems exactly.

Large neighborhood search is an iterative method where we sequentially remove (destruction phase) and reinsert (reconstruction phase) parts of an incumbent solution in the hope of improving it. Neighborhoods defined in the destruction phase are explored in the reconstruction phase. We propose to explore the neighborhoods by column generation managed heuristically. A column generation method is used to solve the linear relaxation of a linear program. It solves iteratively a master problem, that is the linear program restricted to a subset of variables, and one or many subproblems that attempt to find new negative reduced cost variables to add to the master problem. The process ends when the subproblems cannot find any negative reduced cost variables. This method is embedded within a brand-and-bound algorithm to derive integer solutions.

Several operators are defined to select elements that will be removed from the incumbent solution in the destruction phase. At every iteration, an operator is randomly selected favoring those who managed to improve the incumbent solution in the past iterations. Afterwards, column generation is used to explore the neighborhood defined by the operator (reconstruction phase). Many aspects of the column generation approach are managed heuristically in order to obtain good solutions in reasonable time at the expense of ensuring optimality. The subproblems are solved by means of a tabu search algorithm and the column generation is stopped if the value of the solution of the linear relaxation does not improve enough over the last iterations. An aggressive branching scheme is used to derive integer solutions. Branch-

ing is done on the variable with the highest fractional value, which is fixed at 1 without the possibility to backtrack.

This thesis is divided into three main chapters, each corresponding to an article published or submitted for publication. They all present the aforementioned method adapted to a specific vehicle routing problem.

The proposed method is first developed in Chapter 4 for the vehicle routing problem with time windows. The objective is to minimize first the number of vehicles, and then the total distance traveled. That is why the method is adapted in two phases, one for each objective. At time of experimentation, the method managed to improve the best known solutions of 106 over 356 instances of size going from 100 to 1000 customers.

In order to put the method to the test on a more complex problem with real applications, European rules on driver working hours are added to the problem in Chapter 5. Indeed, as of April 2007, an update of the rules governing the driver working hours is in place in the European Union. Even though the rules have a concrete and very practical aspect, very few scientific papers were published on the subject from an optimization point of view. To check the feasibility of a route, a label-setting algorithm is proposed. The driver rules are modeled using complex resource extension functions that can generate multiple labels from one source label. Every time an insertion is evaluated by the tabu search solving the subproblem, the resulting route is checked for feasibility. We also demonstrate how to consider multiple insertions at once by bounding the resource values at each node of the current route of the tabu search. The computed bounds are also used to limit the domain of the label-setting algorithm. The method proposed to check the feasibility of the routes is generic for any algorithm using insertion movements. Compared to two other methods on academic instances, ours is definitely superior.

Finally, the method is generalized in Chapter 6 to a real problem arising in the heating oil distribution industry. Much more complex, the column generation formulation of the problem requires multiple subproblems. In addition, there is a heterogeneous fleet, start and end time of working shifts, multiple depots, intra-route replenishments and optional customers. It is however possible to had a move to the tabu search algorithm solving the subproblems, in order to switch between subproblems without slowing down the method too much. A concurrent tabu search is also presented to solve the whole problem. This method is also inserted into the large neighborhood search algorithm to explore the neighborhoods instead of the heuristic column generation. Computational results show that the large neighborhood search can be a good guiding mechanism for a metaheuristic. However the column generation reconstruction outperforms its tabu search counterpart over the solved instances.

The heuristic column generation method embedded into a large neighborhood search is

therefore is a very competitive approach that can be applied to a wide range of vehicle routing problems. It could be easily adapted to different types of problems where a column generation formulation is applicable. Although slower than other recent approaches, it is possible to obtain better solutions by expanding the size of the neighborhoods in the large neighborhood search.

TABLE DES MATIÈRES

DÉDICACE iii

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT viii

TABLE DES MATIÈRES xi

LISTE DES TABLEAUX xiv

LISTE DES FIGURES xvi

CHAPITRE 1 INTRODUCTION 1
1.1 Problèmes de tournées de véhicules étudiés 3
1.2 Structure de la thèse 5

CHAPITRE 2 REVUE DE LITTÉRATURE 6
2.1 Problème de tournées de véhicules avec fenêtres de temps 6
2.2 Problème de tournées de véhicules avec fenêtres de temps et règles de chauffeurs 7
2.3 Problème de tournées de véhicules pour la livraison d'huile de chauffage 8
2.4 Recherche à grands voisinages 9
2.5 Génération de colonnes 10

CHAPITRE 3 ORGANISATION DE LA THÈSE 15

CHAPITRE 4 A BRANCH-AND-PRICE-BASED LARGE NEIGHBORHOOD SEARCH ALGORITHM FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS 17
4.1 Introduction 18
4.2 Algorithm framework 20
4.3 Destruction 22
4.3.1 Proximity operator 23
4.3.2 Route portion operator 23
4.3.3 Longest detour operator 24

4.3.4 Time operator . . . 25
4.3.5 Roulette-wheel procedure . . . 25
4.4 Reconstruction . . . 26
4.4.1 Heuristic column generation . . . 27
4.4.2 Branching strategy . . . 29
4.5 Computational experiments . . . 30
4.5.1 Instances . . . 30
4.5.2 Parameter values . . . 30
4.5.3 Main results . . . 31
4.5.4 Sensitivity analysis . . . 37
4.6 Conclusion . . . 41
4.7 Nouvelles méthodes . . . 43

CHAPITRE 5 EUROPEAN DRIVER RULES IN VEHICLE ROUTING WITH TIME WINDOWS . . . 45
5.1 Introduction . . . 46
5.2 Problem statement . . . 48
5.2.1 Definitions for driver regulations . . . 49
5.2.2 Regulations on driving time and working time . . . 49
5.2.3 Regulations on rest periods . . . 50
5.3 Solution method . . . 50
5.3.1 Large neighborhood search . . . 51
5.3.2 Branch-and-price heuristic . . . 53
5.3.3 Tabu search column generator . . . 55
5.4 Route feasibility check . . . 55
5.4.1 Labeling algorithm . . . 56
5.4.2 Label extension . . . 57
5.4.3 Approximate feasibility check for multiple insertions . . . 64
5.5 Computational experiments . . . 69
5.6 Conclusion . . . 72
5.7 Appendix A: Additional resources and extension functions . . . 73
5.8 Appendix B: Detailed results . . . 79

CHAPITRE 6 METAHEURISTICS FOR AN OIL DELIVERY VEHICLE ROUTING PROBLEM . . . 81
6.1 Introduction . . . 82
6.2 Problem statement . . . 84

6.3 Optional customer bonuses 86
6.4 Tabu search heuristic 88
6.5 Large neighborhood search heuristic based on tabu search 90
6.6 Large neighborhood search heuristic based on column generation 93
6.7 Computational experiments 96
6.7.1 Single day 96
6.7.2 Week planning 99
6.8 Conclusion 102

CHAPITRE 7 DISCUSSION GÉNÉRALE ET CONCLUSION 104
7.1 Contributions 104
7.2 Avantages et inconvénients de la méthode proposée 105
7.3 Améliorations futures 106

RÉFÉRENCES 109

LISTE DES TABLEAUX

Table 4.1 Parameter values . . . 32
Table 4.2 Solomon's instances with 100 customers . . . 33
Table 4.3 Gehring and Homberger's instances with 200 customers . . . 33
Table 4.4 Gehring and Homberger's instances with 400 customers . . . 34
Table 4.5 Gehring and Homberger's instances with 600 customers . . . 34
Table 4.6 Gehring and Homberger's instances with 800 customers . . . 35
Table 4.7 Gehring and Homberger's instances with 1000 customers . . . 35
Table 4.8 New best solution values . . . 36
Table 4.9 Use of different operators . . . 38
Table 4.10 Statistics on the two phases . . . 38
Table 4.11 Sensitivity analysis on the number of customers removed . . . 40
Table 4.12 Sensitivity analysis on the maximum number of columns in memory pool 40
Table 4.13 Sensitivity analysis on the number of tabu search iterations per column generation iteration in the VNR phase . . . 41
Table 4.14 Sensitivity analysis on the number of tabu search iterations per initial solution in the TDR phase . . . 41
Table 4.15 Sensitivity analysis on the maximum number of VNR iterations to find a feasible solution . . . 42
Table 4.16 Sensitivity analysis on the number of iterations in the TDR phase . . . 42
Tableau 4.17 Nouvelles méthodes pour le VRPTW . . . 43
Table 5.1 Example of resource vectors generated by the label extension function . 65
Table 5.2 Application of Algorithms 5.2 and 5.3 yielding the resource vectors of Table 5.1 . . . 66
Table 5.3 Results for the necessary set of rules of Regulation (EC) No 561/2006 . 72
Table 5.4 Results for the necessary set of rules of Regulation (EC) No 561/2006 and Directive 2002/15/EC . . . 73
Table 5.5 Results for the complete set of rules . . . 74
Table 5.6 Results for the complete set of rules for varying neighborhood sizes . . 74
Table 5.7 Detailed results obtained by the proposed LNS algorithm . . . 80
Table 6.1 Characteristics of the instances created for each date . . . 98
Table 6.2 Results for the single-day instances . . . 99
Table 6.3 Breakdown of the LNS-CG results for the 250-customer instances . . . 100

Table 6.4 LNS-CG results for different numbers of customers removed (250-customer instances) 100
Table 6.5 LNS-CG results for different values of w (one-week instances) 102
Table 6.6 LNS-CG results for different values of μ (one-week instances) 103

LISTE DES FIGURES

Figure 4.1 Algorithm framework 21
Figure 5.1 Algorithm framework 52
Figure 5.2 Three cases for the computation of the current day duration 60
Figure 6.1 Two different detours for customer l 88

CHAPITRE 1

INTRODUCTION

La distribution est une partie importante dans la chaîne d'approvisionnement de nombreuses compagnies. Les entreprises ont des flottes de plus en plus importantes et couvrent des territoires de plus en plus grands. Que ce soit au niveau du marché nord-américain, européen ou ailleurs dans le monde, les coûts engendrés par la distribution sont importants. Pour qu'une compagnie puisse être compétitive, elle doit pouvoir fournir un bon service à moindre coût. Avoir une planification efficace de la distribution permet de diminuer les coûts et d'améliorer le service et donc la satisfaction du client et ainsi se démarquer par rapport à ses compétiteurs. Pour une même demande, cela permet aussi de diminuer l'impact sur l'environnement, un enjeu important du 21e siècle.

Il n'est donc pas surprenant que la recherche de méthodes de planification efficaces pour résoudre les problèmes de distribution suscite autant d'intérêt. Tant d'un point de vue exact que heuristique, un très grand nombre de méthodes ont été développées. L'ensemble des caractéristiques des problèmes abordés est de plus en plus vaste au fur et à mesure que la puissance de calcul des ordinateurs augmente et que les méthodes deviennent plus efficaces. De plus, avec l'avènement du commerce sur internet, les marchés s'élargissent et le nombre de clients augmente. Il est donc important de développer de nouvelles méthodes d'optimisation capables de traiter de façon efficace des problèmes avec un grand nombre de clients tout en considérant un ensemble de contraintes variées.

Le premier problème de tournées de véhicules fut proposé par Dantzig et Ramser (1959) comme une généralisation du célèbre problème du voyageur de commerce. Aujourd'hui mieux connu sous le nom de problème de tournées de véhicules avec capacités, le problème présenté par Dantzig et Ramser consiste à distribuer de la marchandise à un ensemble de clients à partir d'une flotte homogène de véhicules partageant le même dépôt. À ce jour, un grand nombre de variantes ont été proposées et étudiées dans la littérature scientifique. Comme nous nous intéressons aux variantes avec fenêtres de temps, le problème de tournée de véhicules avec fenêtres de temps (VRPTW, pour Vehicle Routing Problem with Time Windows) ainsi que deux variantes plus complexes étudiées sont présentées à la section 1.1.

Dans cette thèse, nous présentons une nouvelle méthode générique pour résoudre une vaste gamme de problèmes de tournées de véhicules. Nous nous intéressons particulièrement aux problèmes comportant des fenêtres de temps, c'est-à-dire que les clients fournissent une plage horaire à l'intérieur de laquelle ils désirent être desservis. L'ajout de cette contrainte

complique grandement le processus de résolution. Par contre, certaines méthodes exactes, notamment la génération de colonnes (voir section 2.5), sont mieux adaptées pour en tenir compte.

Au début des travaux de cette thèse, aucune méthode heuristique tirant profit de la puissance de la génération de colonnes n'avait, à notre connaissance, été développée pour résoudre des problèmes de tournées de véhicules. Une méthode similaire à celle présentée a été appliquée par Pepin *et al.* (2009) pour un problème de construction d'horaires de véhicules. Une méthode générique de génération de colonnes heuristique est aussi présenté par Boschetti *et al.* (2009). Mourgaya et Vanderbeck (2007) sont vraisemblablement les premiers à avoir présenté une méthode pour un problème de tournées de véhicules, soit une variante périodique. Mis à part celle de Pepin *et al.* (2009), aucune de ces méthodes n'insère la génération de colonnes à l'intérieur d'un cadre métaheuristique. Nous nous intéressons donc à une telle méthode afin de résoudre certains problèmes de tournées de véhicules rencontrés dans des contextes réels. Nous nous penchons sur des problèmes avec fenêtres de temps, et à des variantes riches de ces problèmes.

La méthode présentée s'inscrit dans une nouvelle tendance en recherche opérationnelle appelée les matheuristiques (voir Maniezzo *et al.*, 2009). Les matheuristiques combinent les méthodes heuristiques avec des méthodes de programmation mathématique, d'où le nom. Ce type de méthodes existe depuis un certain temps. Puchinger et Raidl (2005) en présentent un survol sans utiliser le terme matheuristique qui a été attribué à ces méthodes plus récemment étant donné l'engouement pour celles-ci. Il existe même depuis 2006 un congrès [1] spécialisée sur le sujet.

On peut séparer ces méthodes en deux catégories principales. Il peut d'abord s'agir d'heuristiques venant aider la résolution du programme mathématique. Par exemple, la résolution du problème de séparation de coupes dans un algorithme de séparation et coupes (branch-and-cut) peut se faire de façon heuristique. Bard *et al.* (2002) utilisent une méthode heuristique pour calculer des bornes supérieures dans une méthode par séparation et évaluation (branch-and-bound). Desaulniers *et al.* (2008) se servent d'une heuristique pour accélérer la résolution du sous-problème de génération de colonnes avant de le résoudre exactement. La deuxième catégorie se compose des méthodes heuristiques à l'intérieur desquelles la programmation mathématique joue un rôle. Archetti *et al.* (2009a) se servent d'un solveur en nombres entiers à l'intérieur d'une recherche tabou pour réorganiser des tournées après un certain nombre d'itérations sans amélioration. Dans cette catégorie, on peut aussi inclure les méthodes où un algorithme exact est utilisé de façon heuristique (Boschetti *et al.*, 2009; Pepin *et al.*, 2009).

La méthode proposée dans cette thèse entre dans la deuxième catégorie. Le mode de

1. http ://astarte.csr.unibo.it/matheuristics

fonctionnement de la méthode exacte est conservé, mais plusieurs étapes sont gérées de façon heuristique. Le tout est inséré dans une méthode métaheuristique de recherche à grands voisinages qui permet de répéter la procédure plusieurs fois sur un espace de solutions plus restreint. De manière similaire, De Franceschi *et al.* (2006) proposent une recherche à grands voisinages pour un problème différent de tournées de véhicules sans fenêtres de temps où la reconstruction (voir section 2.4) est faite par un solveur en nombres entiers.

1.1 Problèmes de tournées de véhicules étudiés

Dans cette thèse, trois problèmes de tournées de véhicules sont abordés. Le premier, le VRPTW, est un problème bien connu qui a beaucoup été étudié. Le second est une variante du VRPTW où un ensemble de règles européennes sur les horaires des chauffeurs sont ajoutées. Le dernier est une variante riche du VRPTW provenant de l'industrie de la distribution d'huile de chauffage.

Considérant une flotte de véhicules associé à un dépôt unique, le VRPTW consiste à déterminer un ensemble de tournées réalisables afin de livrer de la marchandise à un ensemble de clients répartis géographiquement. Chaque client doit être visité une fois par un seul véhicule à l'intérieur d'une fenêtre de temps prescrite afin de satisfaire à sa demande en marchandise. Une tournées est dite réalisable si la somme des demandes des clients visités ne dépasse pas la capacité d'un véhicule et que les clients sont visités à l'intérieur de leur fenêtre de temps. L'objectif visé est de minimiser d'abord le nombre de véhicules utilisés et ensuite la distance totale parcourue. Dans certains cas, le premier de ces objectifs est omis.

Le second problème incorpore au VRPTW des règles qui doivent être respectées par les horaires des chauffeurs pour obtenir le problème de tournées de véhicules avec fenêtre de temps et règles de chauffeurs. Les tournées créées doivent donc pouvoir être réalisées par un chauffeur soumis à un ensemble de règles complexes sur son horaire de travail. Les règles considérées sont celles imposées par l'union européenne en avril 2007 dans la législation (EC) No 561/2006 (European Union, 2006). Ces règles doivent être respectées par tous les chauffeurs sous peine d'amende. Il est donc impératif pour les compagnies de distribution de donner des charges de travail réalisables pour les chauffeurs. Une explication détaillée des règles se trouve à la section 5.2.1. En résumé, elles limitent les heures de conduite d'un chauffeur. Il doit régulièrement prendre des périodes de pause et de plus longues périodes de repos quotidiennement et hebdomadairement. Les règles imposent des limites sur le nombre d'heures conduites sans prendre de pause ou de repos. Les heures de conduite ne doivent pas être nécessairement consécutives, elles peuvent être, par exemple, entrecoupées par du travail autre que de la conduite. De plus, il existe un grand nombre d'exceptions permettant

de déroger aux règles précédentes dans certaines conditions. Les chauffeurs doivent aussi se conformer à des règles plus générales sur les heures de travail définies par la directive 2002/15/EC (European Union, 2002). Celles-ci imposent un maximum d'heures de travail sans pause et un maximum d'heures de travail pour une période d'une semaine.

Le dernier problème étudié vient du domaine de la distribution d'huile de chauffage. Chaque jour, les distributeurs doivent remplir le réservoir d'huile de chauffage d'un ensemble de clients. L'inventaire des clients est souvent géré par le vendeur et est estimé grâce à l'historique de consommation du client et aux températures moyennes depuis le dernier ravitaillement. Chaque client possède une quantité idéale de livraison définie par le distributeur. Si l'estimation de la consommation du client dépasse cette quantité, il y a un risque de vider le réservoir, ce qui peut être très couteux. Si l'estimation est plus petite et qu'on dessert le client, on devra le ravitailler à nouveau plus tôt, ce qui n'est pas idéal.

À chaque jour, un ensemble de clients doivent être desservis car l'estimation de leur consommation a atteint la quantité idéale de livraison. De plus, il peut être bénéfique de desservir des clients qui n'ont pas tout à fait atteint la quantité idéale de livraison mais qui sont proches de ceux qui seront desservis. Étant donné la nature incertaine de l'évolution de la demande des clients, nous considérons le problème sur un horizon d'une seule journée qui doit être résolu la veille des opérations. Le problème est formulé à la manière d'un problème de tournées de véhicules où, pour une journée donnée, un ensemble de clients obligatoires doivent être desservis ainsi qu'un ensemble de clients optionnels. Les clients obligatoires ont soit atteint la quantité idéale de livraison ou bien leur inventaire est géré différemment (ils peuvent par exemple être sur appel et doivent eux-mêmes contacter le distributeur lorsque leur réservoir est presque vide). Les clients optionnels sont ceux qui seraient obligatoires dans les jours qui viennent mais il peut être souhaitable de les desservir plus tôt si cela évite un détour. Pour chaque client optionnel, il faut évaluer une valeur de 'bonus' qui est une estimation du gain encourue en le desservant à l'avance. Certains clients possèdent une fenêtre de temps à l'intérieur de laquelle ils doivent être visités.

Le problème consiste donc à effectuer la livraison à un ensemble de clients obligatoires ainsi qu'à des clients optionnels si un gain peut être encouru. Pour cela, on dispose d'une flotte hétérogène de véhicules avec des capacités différentes, chacun pré-associé à un chauffeur ayant un certain horaire de travail (heure de début et de fin). Chaque tournée doit donc respecter l'horaire du chauffeur ainsi que la capacité du véhicule qui lui est associée. Les véhicules sont affectés à plusieurs dépôts et ont la possibilité de se ravitailler en cours de route à un ensemble de points de ravitaillement. Le problème, appelé problème de tournées de véhicules pour la livraison d'huile, est donc une généralisation du VRPTW avec plusieurs dépôts, une flotte hétérogène, des quarts de travail, des ravitaillements intra-routes et des clients optionnels.

1.2 Structure de la thèse

Le présent document est structuré comme suit. D'abord, le chapitre 2 propose une revue critique de la littérature sur les problèmes et méthodes qui nous intéressent. Le chapitre 3 présente ensuite l'organisation des travaux de recherche ayant mené à cette thèse. Les chapitres 4 à 6 contiennent les articles scientifiques publiés (ou en voie de publication) découlant de cette recherche. Vient ensuite une discussion générale sur les résultats obtenus et une conclusion (chapitre 7).

CHAPITRE 2

REVUE DE LITTÉRATURE

Ce chapitre présente un aperçu des articles concernant les problèmes étudiés et les principales méthodologies utilisées dans cette thèse. Une brève revue de littérature sur les problèmes abordés est d'abord présentée aux sections 2.1 à 2.3. Ensuite la section 2.4 survole les différentes applications de recherche à grands voisinages (LNS) puis la section 2.5 aborde les méthodes de génération de colonnes appliquées à des problèmes de tournées de véhicules.

2.1 Problème de tournées de véhicules avec fenêtres de temps

Un très grand nombre de méthodes heuristiques ont été appliquées afin de résoudre le VRPTW. Les toutes premières méthodes ont tenté de construire des tournées de façon gloutonne. Par la suite, des opérateurs de recherche locale ont été développés pour aboutir finalement à un nombre important de méthodes métaheuristiques variées. Le lecteur est prié de se référer à Bräysy et Gendreau (2005a,b) pour une synthèse exhaustive des méthodes heuristiques et métaheuristiques qui ont été appliquées au VRPTW.

Clarke et Wright (1964) ont proposé un algorithme ('savings heuristic') où les tournées sont construites séquentiellement en insérant un à un des clients qui maximisent une 'mesure d'économie'. Solomon (1987) a proposé une extension de cette méthode, tout comme Ioannou *et al.* (2001) et plusieurs autres. Potvin et Rousseau (1993) ont développé une version de cet algorithme où les tournées sont construites parallèlement.

La recherche s'est vite tournée vers des méthodes itératives pouvant améliorer des solutions obtenues par un algorithme de construction de tournées. La plupart de ces méthodes sont des algorithmes d'échanges d'arcs adaptés à partir de ceux introduits originalement pour le problème du voyageur de commerce. Les voisinages d'échange d'arcs sont formés en retirant k arcs de la solution courante et en les remplaçant par k nouveaux arcs. On appelle ces voisinages k-opt ou k-exchange. On compte aussi parmi les opérateurs les plus populaires, les *relocate* et *exchange* (Savelsbergh, 1992), le *CROSS-exchange* (Taillard *et al.*, 1997), le *GENI-exchange* (Gendreau *et al.*, 1992) et les chaînes d'éjection (Glover, 1991, 1992).

Du côté métaheuristique, un grand éventail de méthodes ont été proposées dont la recherche tabou (Chiang et Russell, 1997; Taillard *et al.*, 1997; Cordeau *et al.*, 2001b) où une ou plusieurs structures de voisinages bien connues sont utilisées. Lim et Zhang (2007) ont aussi appliqué un algorithme d'ascension de colline (hill-climbing algorithm).

De plus, beaucoup de méthodes évolutives ont été appliquées au VRPTW. La plupart des travaux sur ce sujet présentent une méthode hybride avec différents algorithmes de construction de tournées, de recherche locale et même d'autres métaheuristiques. Selon Bräysy et Gendreau (2005b), les méthodes évolutives de Homberger et Gehring (2005) et Berger *et al.* (2003) sont parmi les plus efficaces. Mester et Bräysy (2005), Hashimoto et Yagiura (2008), Repoussis *et al.* (2009) et Nagata *et al.* (2010) ont développé plus récemment des méthodes hybrides très compétitives combinant la recherche locale avec une méthode évolutive.

De nombreuses autres métaheuristiques ont été appliquées au VRPTW telles que la recherche à voisinages variables (Rousseau *et al.*, 2002; Bräysy, 2003), la recherche à grands voisinages (Shaw, 1998; Pisinger et Ropke, 2007), le recuit simulé (Bent et Van Hentenryck, 2004), des algorithmes de recherche locale itérative (Ibaraki *et al.*, 2005, 2008), des méthodes hybrides combinant plusieurs métaheuristiques (Bent et Van Hentenryck, 2004), et même une méthode de recherche parallèle coopérative exploitant plusieurs métaheuristiques connues (Le Bouthillier *et al.*, 2005).

Du côté exact, les méthodes les plus efficaces sont celles de génération de colonnes. Une revue de ces méthodes se trouve à la section 2.5. Quelques méthodes de plans coupants (branch-and-cut) ont aussi été développées dans les dernières années. Celles-ci utilisent un grand nombre d'inégalités tels que l'élimination de sous-tours, les coupes 2-chemins, etc. Bard *et al.* (2002) utilisent en plus une procédure de recherche adaptative gloutonne aléatoire (GRASP) afin de calculer une borne supérieure. Les autres méthodes à noter sont celles de Lysgaard (2006) et Kallehauge *et al.* (2007). Ce type de méthodes ne semble toutefois pas compétitif avec les méthodes de génération de colonnes.

2.2 Problème de tournées de véhicules avec fenêtres de temps et règles de chauffeurs

À ce jour, très peu d'articles ont traité des règles sur les horaires de chauffeurs en cours dans l'union européenne (European Union, 2006). Les méthodes proposées dans la littérature utilisent presque toutes des algorithmes d'étiquetage à l'intérieur desquels des fonctions de prolongation de ressources modélisent les règles. Goel (2009) a présenté la première méthode où il ne considère qu'un sous-ensemble des règles. Il propose des règles de dominance ainsi qu'un ensemble d'instances basées sur celles de Solomon (1987). Il intègre sa méthode à l'intérieur d'une recherche à grands voisinages relativement simple. Dans Goel (2010), les fonctions de prolongation et règles de dominance présentées dans Goel (2009) sont généralisées à l'ensemble des règles européennes. Kok *et al.* (2010) proposent aussi des fonctions de prolongation, cette fois heuristiques, à l'intérieur d'un algorithme de programmation dyna-

mique qui considère tout le problème d'un seul coup. Ils démontrent que certaines extensions des règles, bien que non nécessaires, permettent d'améliorer la qualité des solutions si on les considère. Drexl et Prescott-Gagnon (2010) présentent un algorithme d'étiquetage pour un problème de plus court chemin avec contraintes de ressources. Finalement, Zäpfel et Bögl (2008) présentent un algorithme à deux phases pour résoudre un problème de tournées de véhicules complexe avec quelques-unes des règles européennes.

Vérifier si une séquence de clients peuvent être visités par une tournées satisfaisant un ensemble de règles d'horaires de chauffeurs est un problème combinatoire en soi. Archetti et Savelsbergh (2009) proposent un algorithme pour les règles en cours aux États-Unis. Ils démontrent que, sous certaines conditions, il est possible de trouver un horaire satisfaisant ces règles, ou de prouver qu'il n'en existe pas, en un temps de l'ordre de $O(n^3)$, où n est le nombre de clients sur la tournée. Goel et Kok (2009) ont ensuite présenté un algorithme en $O(n^2)$ pour les même règles.

Des règles similaires, quoi que beaucoup plus simples, ont été traitées dans d'autres articles. Savelsbergh et Sol (1998) abordent comment traiter des pauses de dîner dans un algorithme de génération de colonnes pour un problème de tournées de véhicules. Xu *et al.* (2003) considèrent, entre autres, des limites d'heures de conduite par jour. Finalement, Bartodziej *et al.* (2009) présentent plusieurs méthodes pour résoudre un problème combiné de tournées de véhicules et d'horaires de travail avec des contraintes de pause.

2.3 Problème de tournées de véhicules pour la livraison d'huile de chauffage

Le problème de livraison d'huile de chauffage abordé dans cette thèse est très spécifique. À notre connaissance, il y a pas d'autre article portant exactement sur la même problématique. Par contre certains aspects ont déjà été traités. Dror (2005) présente un survol des méthodes portant sur la distribution de propane (équivalent à de l'huile de chauffage). Plusieurs problèmes semblables (Dror *et al.*, 1985; Dror et Ball, 1987) sont modélisés comme des problèmes combinés de tournées de véhicules et de gestion d'inventaire (IRP pour Inventory Routing Problems). Plus récemment, les recherches se sont concentrées sur des variantes de IRP où les demandes sont stochastiques (Kleywegt *et al.*, 2002; Adelman, 2004). Il existe aussi des similarités avec des problèmes de tournées de véhicules multi-périodiques. Une méthode de génération de colonnes a, entre autres, été développée par Bostel *et al.* (2008). Des caractéristiques particulières de notre problème tels que des dépôts multiples (Cordeau *et al.*, 1997; Pisinger et Ropke, 2007) ou des ravitaillements intra-routes (Angelelli et Speranza, 2002; Tarantilis *et al.*, 2008; Crevier *et al.*, 2007) ont aussi été traitées dans la littérature. Plus d'explications sur ces méthodes sont données à la section 6.1.

2.4 Recherche à grands voisinages

La recherche à grands voisinages (LNS, pour large neighborhood search) est une méthode heuristique itérative qui, à chaque itération, retire une partie des éléments d'une solution (destruction) avant de les réinsérer différemment (reconstruction) dans l'espoir d'améliorer la solution courante. Un voisinage est donc défini par l'ensemble des éléments qui n'ont pas été retirés de la solution courante. Puisque la taille du voisinage augmente de façon exponentielle en fonction du nombre d'éléments retirés, il est possible de modifier de façon significative la solution courante, d'où le nom de recherche à grands voisinages. Il y a deux aspects importants à définir pour un tel algorithme. Le premier est la méthode de destruction de la solution courante. Celle-ci est inhérente à la structure du problème. Le nombre d'éléments à retirer est aussi à considérer dans cet aspect. Il doit être assez grand pour assurer un changement considérable de la solution courante bien qu'un nombre trop grand ralentisse considérablement les temps de calcul. Le second aspect est la méthode de reconstruction. Celle-ci doit être assez puissante pour permettre de trouver une bonne solution mais suffisamment rapide pour permettre à l'algorithme de faire un nombre important d'itérations dans un temps raisonnable.

L'algorithme 2.1 présente un pseudo-code d'une méthode LNS générique. Il faut d'abord une solution initiale Sol_0 et un paramètre TotLnsIter indiquant le nombre maximal d'itérations de la méthode. Le processus itératif peut ensuite commencer. Si la méthode a plusieurs opérateurs de destruction, il faut d'abord en choisir un. Pisinger et Ropke (2007) propose une méthode aléatoire favorisant les opérateurs ayant eu un meilleur taux de réussite dans les itérations précédentes. L'opérateur est ensuite appliqué pour détruire la solution courante Sol_i, dans la fonction Détruire(). Si plusieurs opérateurs de reconstruction existent, on en choisit un et on reconstruit la solution courante partielle $\overline{\text{Sol}}_{i+1}$ détruite grâce à la fonction Reconstruire() pour obtenir une nouvelle solution Sol_{i+1}. Le tout est répété pour un nombre fixé d'itérations.

Shaw (1998) a été vraisemblablement le premier à proposer un algorithme LNS et ce afin de résoudre le VRPTW. Il définit un seul opérateur de destruction basé sur une valeur de proximité entre les éléments de la solution et la programmation par contraintes est utilisée pour la reconstruction. D'autres chercheurs ont par la suite apporté des modifications à cette méthode, par exemple en définissant de nouveaux opérateurs de destruction (Rousseau *et al.*, 2002) ou en utilisant un LNS à l'intérieur d'une méthode hybride (Bent et Van Hentenryck, 2004). Un LNS a aussi été appliqué sur plusieurs autres problèmes de tournées de véhicules (Pisinger et Ropke, 2007; Goel et Gruhn, 2005).

Plusieurs travaux ont porté sur d'autres problèmes tels que des problèmes d'ordonnance-

Algorithme 2.1 Recherche à grands voisinages

Entrée: Sol_0, solution initiale
Entrée: TotLnsIter, nombre total d'itérations
  $i \leftarrow 0$
  MeilleureSolution $\leftarrow \text{Sol}_0$
  répéter
    OpDestruct $\leftarrow$ ChoisirOperateurDestruction()
    $\overline{\text{Sol}}_{i+1} \leftarrow$ Detruire(Sol_i,OpDestruct)
    OpReconstruct $\leftarrow$ ChoisirOperateurReconstruction()
    $\text{Sol}_{i+1} \leftarrow$ Reconstruire($\overline{\text{Sol}}_{i+1}$,OpReconstruct)
    si Sol_{i+1}.coût < MeilleureSolution.coût **alors**
      MeilleureSolution $\leftarrow \text{Sol}_{i+1}$
    $i \leftarrow i + 1$
  jusqu'à $i \geq$ TotLnsIter
  Retourner MeilleureSolution

ment (Carchrae et Beck, 2005; Godard *et al.*, 2005; Laborie et Godard, 2007). Finalement, notons que Perron *et al.* (2004) ont défini un algorithme LNS général pour résoudre des problèmes de programmation par contraintes, où la propagation des contraintes sert à définir les voisinages.

Finalement, plusieurs autres méthodes proposées dans la littérature peuvent s'apparenter à un LNS. Bien avant, les techniques de 'shuffling' pour les problèmes d'ordonnancement de 'job-shop' présentent les mêmes caractéristiques (Applegate et Cook, 1991), Shaw (1998) les donnant même en exemple. L'idée de base est aussi globalement la même que la méthode 'ruin and recreate' de Schrimpf *et al.* (2000), ou celle de Caseau et Laburthe (1999) intitulée 'Forget and extend'.

2.5 Génération de colonnes

La génération de colonnes est une méthode itérative utilisée pour résoudre des programmes linéaires. Au lieu de considérer le programme linéaire dans son ensemble, on le décompose en un problème maître restreint (PMR) et en un ou plusieurs sous-problèmes. Le PMR est tout simplement le programme linéaire initial restreint à un sous-ensemble de variables. Les sous-problèmes servent à générer de nouvelles variables pour le PMR. Ces sous-problèmes doivent avoir la propriété de générer de nouvelles variables si celles-ci peuvent améliorer la solution du PMR et de s'assurer qu'il n'y a aucune variable pouvant améliorer la solution du PMR, le cas échéant. Il peut être utile d'appliquer une telle technique lorsque le nombre de variables est trop grand pour qu'une énumération explicite soit réaliste.

Afin de pouvoir appliquer une méthode de génération de colonnes à un problème, il faut d'abord trouver une formulation qui satisfait les conditions énumérées ci-haut. Heureusement,

c'est le cas pour un grand nombre de problèmes de tournées de véhicules. On peut définir un problème générique de tournées de véhicules où un ensemble de clients peuvent être desservis par des véhicules sur des tournées de différents types. Les différents types de tournées peuvent être dus à des dépôts différents, à des capacités de véhicules différentes, à des horaires de travail différents, etc.

On peut alors définir les variables et paramètres suivants. Soit Ω^t, $t \in \mathcal{T}$, l'ensemble des tournées réalisables d'un même type t. À chaque tournée $p \in \Omega^t$, $t \in \mathcal{T}$, sont associés les paramètres suivants : c_p^t, son coût et v_{ip}^t, $i \in \mathcal{N}$, prenant la valeur 1 si le client i est visité par la tournée p et 0 sinon. On peut définir une valeur $\mathcal{K}^t, t \in \mathcal{T}$ comme étant le nombre maximal de tournées du type t. Finalement, une variable binaire θ_p^t est définie pour chaque tournée $p \in \Omega^t$, $t \in \mathcal{T}$, prenant la valeur 1 si la tournée p fait partie de la solution et 0 sinon.

Un problème de tournées de véhicules peut alors se formuler comme un problème de partitionnement d'ensemble avec contraintes supplémentaires :

$$\text{Minimiser} \qquad \sum_{t\in\mathcal{T}} \sum_{p\in\Omega^t} c_p^t \theta_p^t \tag{2.1}$$

$$\text{sujet à :} \qquad \sum_{t\in\mathcal{T}} \sum_{p\in\Omega^t} v_{ip}^t \theta_p^t = 1, \qquad \forall\, i \in \mathcal{N} \tag{2.2}$$

$$\sum_{p\in\Omega^t} \theta_p^t \leq \mathcal{K}^t, \qquad \forall\, t \in \mathcal{T} \tag{2.3}$$

$$\theta_p^t \in \{0,1\}, \qquad \forall\, p \in \Omega^t, \forall\, t \in \mathcal{T}. \tag{2.4}$$

La fonction objectif (2.1) vise à minimiser le coût total. Les contraintes (2.2) assurent que chaque client ne soit visité qu'une seule fois sur une seule tournée. Les contraintes (2.3) limitent, si nécessaire, le nombre de tournées de chaque type t. À partir de cette formulation, il est possible d'appliquer une méthode de génération de colonnes pour résoudre le problème. Le nombre de tournées réalisables peut être extrêmement grand et une énumération explicite n'est pas réaliste.

Bien que cette formulation englobe un grand nombre de variantes de problèmes de tournées de véhicules, elle ne couvre pas l'ensemble des possibilités. Par exemple, les clients optionnels définis dans le problème de livraison d'huile de chauffage présenté à la section 1.1 ne sont pas couverts par la formulation. Le modèle (2.1)–(2.4) a été choisi par souci de concision. D'autres contraintes doivent être ajoutées au modèle pour les prendre en compte.

La génération de colonnes est utilisée pour résoudre la relaxation linéaire du modèle (2.1)–(2.4) qui est le problème maître. On résout itérativement le problème maître restreint (PMR)

à un sous-ensemble de variables et plusieurs sous-problèmes, soit un par type de tournée t. La résolution du PMR fournit une solution primale et une solution duale. Cette solution duale est transférée aux sous-problèmes dont l'objectif est de générer des variables de coût réduit négatif à rajouter au PMR. Ce dernier est alors résolu de nouveau à la prochaine itération avec le sous-ensemble augmenté de variables. La méthode se termine quand aucun sous-problème ne peut générer une variable de coût réduit négatif. On peut alors conclure que la solution du PMR est optimale.

Le PMR peut être résolu grâce à un algorithme de programmation linéaire tel que la méthode du simplexe primal. Pour des problèmes de tournées de véhicules, les sous-problèmes (voir Irnich et Desaulniers, 2005) sont des problèmes de plus court chemin élémentaire avec contraintes de ressources (ESPPRC, pour Elementary Shortest Path Problem with Resource Contraints) définis pour chaque type de tournée t. Ce type de sous-problème est NP-difficile (Dror, 1994) et peut être résolu de façon exacte par des algorithmes de programmation dynamique.

À titre d'exemple, il y a un ESPPRC par dépôt dans un problème de tournées de véhicules avec dépôts multiples. Dans le modèle (2.1)–(2.4), chaque type de tournée t est associé aux tournées partant d'un des dépôts. Chaque ESPPRC peut donc être représenté sur un réseau $G^t = (\mathcal{V}^t, \mathcal{A}^t)$ pour chaque type de tournée t. À chaque client $i \in \mathcal{N}$ sont associées une demande q_i et possiblement une fenêtre de temps $[a_i, b_i]$ à l'intérieur de laquelle il doit être visité. Chaque ensemble $\mathcal{V}^t, t \in \mathcal{T}$, contient un nœud pour chaque client $i \in \mathcal{N}^t$ qui peut être visité par une tournée du type t ainsi qu'un nœud source $o(t)$ et un nœud puits $d(t)$, représentant les points de départ et d'arrivée des tournées de type t. L'ensemble $\mathcal{A}^t, t \in \mathcal{T}$ contient les arcs sortant du point de départ $(o(t), j)$, $\forall\, j \in \mathcal{N}^t$, entrant au point d'arrivée $(j, d(t)), \forall\, j \in \mathcal{N}$, et des arcs de déplacement (i, j), $\forall\, i, j \in \mathcal{N}^t$, si les tournées de type t peuvent se déplacer entre i et j et, dans le cas de fenêtres de temps, si le client j peut être visité tout de suite après le client i dans au moins une tournée possible, c'est-à-dire si $a_i + t_{ij} \leq b_j$, où t_{ij} correspond au temps de déplacement entre i et j plus un temps de service, le cas échéant. À chaque arc $(i, j) \in \mathcal{A}^t$, $t \in \mathcal{T}$ est associé un coût de déplacement c_{ij}. À noter qu'il est possible d'avoir des réseaux plus complexes pour d'autres types de problèmes de tournées de véhicules.

Par contre, pour calculer le chemin de plus petit coût réduit lors de la résolution du sous-problème, les coûts c_{ij}, $(i, j) \in \mathcal{A}^t$, sont remplacés par des coûts réduits

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \pi_i & \text{si } i \in \mathcal{N}^t \\ c_{ij} - \mu_t & \text{si } i = o(t), \end{cases}$$

où π_i, $i \in \mathcal{N}^t, t \in \mathcal{T}$, et $\mu_t, t \in \mathcal{T}$ sont les valeurs des variables duales des contraintes (2.2) et

(2.3) du PMR à l'itération courante, respectivement.

Chaque tournée réalisable peut être représentée par un chemin dans G^t. Par contre, des contraintes de ressources sont requises pour assurer la faisabilité des chemins. Une ressource peut être ajoutée pour le respect des fenêtres de temps, une pour le chargement des véhicules, une ressource de visite par client pour assurer l'élémentarité, etc. Une ressource est une quantité qui s'accumule le long d'un chemin et qui est restreinte à un intervalle à chaque nœud. Il est possible de modéliser plusieurs types de contraintes par des ressources (voir Irnich et Desaulniers, 2005).

Afin d'obtenir des solutions entières, il est possible d'imbriquer la méthode de génération de colonnes dans une méthode de séparation et évaluation, appelée en anglais *branch-and-bound*. La méthode porte alors le nom de *branch-and-price* en anglais (Barnhart *et al.*, 1998b; Desaulniers *et al.*, 1998a). Différentes décisions de branchement peuvent être imposées. Par exemple des décisions sur le flot sur les arcs peuvent être imposées dans le sous-problème ou sur le nombre de véhicules utilisés par dépôt, imposées dans le problème maître. Il est toutefois difficile de brancher directement sur les variables du PMR, car pour imposer $\theta_t^p = 0$, on doit interdire la génération de la tournée p dans le sous-problème associé aux tournées de type t. Toutefois, on peut le faire dans un contexte heuristique où la branche $\theta_t^p = 0$ n'est pas explorée.

La génération de colonnes est apparue au début des années 1960 à partir du principe de décomposition de Dantzig et Wolfe (1960) pour un programme linéaire quelconque et aussi directement pour le problème de découpe (Gilmore et Gomory, 1961). Depuis, il y a eu plusieurs articles de synthèse sur le sujet (Desrosiers *et al.*, 1995; Desaulniers *et al.*, 1998a; Barnhart *et al.*, 1998b; Lübbecke et Desrosiers, 2005) de même qu'un livre (Desaulniers *et al.*, 2005).

Desrochers *et al.* (1992) ont probablement été les premiers à appliquer une méthode de génération de colonnes efficace pour le VRPTW. Afin de simplifier le ESPPRC, une relaxation est obtenue en omettant les contraintes d'élémentarité (SPPRC). On peut ainsi obtenir des chemins contenant des cycles. Grâce à une procédure éliminant les 2-cycles, ils ont pu obtenir de meilleures bornes inférieures.

La suite de la recherche sur le VRPTW s'est orientée dans deux directions principales pour tenter d'améliorer la qualité des bornes inférieures, soit l'ajout de plans coupants (branch-and-price-and-cut) et l'enrichissement du sous-problème pour éviter les cycles.

Dans cette première direction, Kohl *et al.* (1999) ont introduit les coupes k-chemins. Ils ont testé seulement les coupes 2-chemins mais avec beaucoup de succès. Cook et Rich (1999) ont démontré que l'application des coupes k-chemins pour $k \geq 3$ peut augmenter substantiellement les bornes inférieures. Jepsen *et al.* (2008) ont appliqué des coupes de

Chvátal-Gomory définies sur les variables du problème maître et ont démontré comment modifier le sous-problème pour en tenir compte.

Au niveau du sous-problème, plusieurs stratégies ont été élaborées. Irnich et Villeneuve (2006) ont développé une procédure d'élimination de k-cycles pour le SPPRC pour des valeurs arbitraires de k. Plusieurs travaux ont aussi porté sur le ESPPRC (Feillet *et al.*, 2004, 2007; Chabrier, 2006; Righini et Salani, 2006), où des ressources sont ajoutées pour assurer l'élémentarité. Righini et Salani (2008) et Boland *et al.* (2006) ont développé indépendamment des algorithmes semblables pour le ESPPRC où les ressources sont ajoutées dynamiquement si des tournées trouvées ne sont pas élémentaires.

Une autre approche utilisée plus récemment sur le VRPTW pour accélérer la génération de colonnes est la collaboration avec des méthodes heuristiques. Danna et Le Pape (2005) ont proposé une méthode coopérative entre la génération de colonnes, un solveur en nombre entiers (basé sur une méthode de séparation et évaluation) et une métaheuristique (recherche tabou ou LNS, par exemple) afin d'obtenir des solutions entières tout au long du processus de résolution. Desaulniers *et al.* (2008) ont introduit une méthode tabou pour résoudre le ESPPRC en début de résolution par génération de colonnes quand beaucoup de colonnes de coût réduit négatif existent. Ils ont aussi développé comme sous-problème un problème de plus court chemin partiellement élémentaire avec contraintes de ressources dans lequel l'élémentarité est seulement imposée sur un sous-ensemble de nœuds qui sont choisis dynamiquement.

La génération de colonnes a aussi été appliquée pour de nombreux autres problèmes de tournées de véhicules, notamment, avec cueillettes et livraisons (Dumas *et al.*, 1991; Savelsbergh et Sol, 1995; Ropke et Cordeau, 2009; Gutiérrez-Jarpa *et al.*, 2010), avec livraisons partagées (Gendreau *et al.*, 2006; Desaulniers, 2010; Archetti *et al.*, 2009b) et avec demandes stochastiques (Christiansen et Lysgaard, 2007). Elle est aussi couramment utilisée dans d'autres domaines du transport pour résoudre des problèmes d'horaires de véhicules et d'horaires de membres d'équipages, entre autres, en transport urbain (Ribeiro et Soumis, 1994; Desaulniers *et al.*, 1998b; Desrochers et Soumis, 1989; Haase *et al.*, 2001), en transport aérien (Desaulniers *et al.*, 1997; Barnhart *et al.*, 1998a; Vance *et al.*, 1997; Gamache *et al.*, 1999; Klabjan *et al.*, 2002; Boubaker *et al.*, 2010), en transport ferroviaire (Ziarati *et al.*, 1997; Cordeau *et al.*, 2001a; Peeters et Kroon, 2008; Huisman, 2007) et en transport maritime (Christiansen et Nygreen, 2006; Gronhaug *et al.*, 2010). On retrouve aussi des applications de cette méthode dans divers autres domaines, notamment, en production et en télécommunications.

CHAPITRE 3

ORGANISATION DE LA THÈSE

Parmi les méthodes exactes les plus efficaces pour résoudre des problèmes de tournées de véhicules se trouve la génération de colonnes. Par contre, une des instances bien connues du VRPTW avec seulement 100 clients demeure à ce jour non résolue de façon exacte (voir Baldacci *et al.*, 2010). L'objectif de cette thèse est donc de développer une méthode tirant profit de la puissance de la génération de colonnes pour résoudre des problèmes de tournées de véhicules de plus grande taille. Du côté métaheuristique, le LNS a obtenu de bons résultats sur des problèmes de tournées de véhicules. L'intégration du LNS avec la génération de colonnes étant assez naturelle, cette métaheuristique fournit un cadre idéal pour rendre heuristique la génération de colonnes et ainsi pouvoir s'attaquer à des instances de problèmes de tournées de véhicules de grande taille. Afin de pouvoir vérifier l'application de la méthode pour des problèmes réels, nous nous intéressons aussi à des variantes riches provenant d'applications réelles. La principale contribution de cette thèse est donc d'insérer la génération colonnes à l'intérieur d'un LNS, et ce, pour résoudre un ensemble de problèmes de tournées de véhicules. La méthode ainsi développée s'inscrit dans un nouveau courant de recherche appelé les matheuristiques, présenté en introduction. Cette thèse se divise en trois grandes phases.

La première consiste à développer la méthode hybride heuristique de génération de colonnes pour le VRPTW. Ce problème est assez simple et pertinent pour que des dizaines de méthodes y aient été appliquées (voir section 2.1) mais suffisamment complexe pour qu'il y ait toujours matière à amélioration. C'est donc un problème idéal pour tester une nouvelle méthode pour des problèmes de tournées de véhicules. Dans cette phase, constituant le chapitre 4, on définit le cadre du LNS à l'intérieur duquel la génération de colonnes vient explorer les voisinages. Les stratégies utilisées pour rendre la génération de colonnes heuristique sont aussi exposées. Plusieurs étapes de la génération de colonnes sont gérées de façon heuristique, de la résolution du sous-problème, à la méthode de branchement pour obtenir des solutions réalisables. De plus, on peut parfois considérer deux objectifs pour le VRPTW, soit réduire le nombre de véhicules puis réduire la distance parcourue pour le nombre minimal de véhicules. Nous présentons donc comment adapter la méthode pour pouvoir réduire le nombre de véhicules et ensuite la distance parcourue. Les résultats obtenus sur les instances connues démontrent la validité de la méthode.

La deuxième phase présentée au chapitre 5 consiste à généraliser la méthode pour un problème plus complexe qui est une généralisation du VRPTW où le sous-problème est gran-

dement enrichi. Tel que présenté à la section 1.1, le problème consiste à faire en sorte que les tournées créées puissent satisfaire un ensemble de règles complexes sur les horaires de chauffeurs. Les règles étant issues de l'union européenne, on s'attaque à un problème dont la portée réelle est clairement définie. On démontre comment gérer ces règles en les modélisant à l'aide de fonctions de prolongation de ressources. Un algorithme d'étiquetage (label-setting) utilisant ces fonctions pour vérifier de façon heuristique la faisabilité des tournées est inséré à l'intérieur de la méthode présentée dans la première phase. Les propriétés des fonctions de prolongation sont aussi utilisées afin de restreindre le voisinage à visiter par la méthode tabou résolvant le sous-problème.

La dernière phase (chapitre 6) consiste à généraliser à nouveau la méthode mais pour un problème de distribution d'huile de chauffage provenant de l'industrie. Ce problème, présenté à la section 1.1, est un problème réel où la taille des instances peut être très grande, ce qui rejoint l'objectif principal de cette thèse. Ce problème enrichit la formulation de génération de colonnes à la fois au niveau du problème maître ainsi que du sous-problème. La méthode doit donc être adaptée pour tenir compte de ces modifications. De plus, la structure du problème engendre un ensemble de sous-problèmes différents. Une méthode tabou concurrente est aussi développée afin de proposer une alternative à la méthode de génération de colonnes heuristique. Cette méthode permet aussi de démontrer la validité du LNS comme mécanisme de guidage et d'accélération pour une autre métaheuristique.

CHAPITRE 4

A BRANCH-AND-PRICE-BASED LARGE NEIGHBORHOOD SEARCH ALGORITHM FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

Article publié dans *Networks* 54(4), 190–204, 2009, et écrit par:

ERIC PRESCOTT-GAGNON

École Polytechnique de Montréal

GUY DESAULNIERS

École Polytechnique de Montréal

LOUIS-MARTIN ROUSSEAU

École Polytechnique de Montréal

Abstract

Given a fleet of vehicles assigned to a single depot, the vehicle routing problem with time windows (VRPTW) consists of determining a set of feasible vehicle routes to deliver goods to a set of customers while minimizing, first, the number of vehicles used and, second, total distance traveled. A large number of heuristic approaches for the VRPTW have been proposed in the literature. In this paper, we present a large neighborhood search algorithm that takes advantage of the power of branch-and-price which is the leading methodology for the exact solution of the VRPTW. To ensure diversification during the search, this approach uses different procedures for defining the neighborhood explored at each iteration. Computational results on the Solomon's and the Gehring and Homberger's benchmark instances are reported. Compared to the best known methods, the proposed algorithm produces better solutions, especially on the largest instances where the number of vehicles used is significantly reduced.

Keywords: Vehicle routing, time windows, large neighborhood search, heuristic column generation.

4.1 Introduction

Given a fleet of vehicles assigned to a single depot, the vehicle routing problem with time windows (VRPTW) consists of determining a set of feasible routes (one route per vehicle used) to deliver goods to a set $\mathcal{N}$ of scattered customers while minimizing, first, the number of vehicles used and, second, the total distance traveled (which is usually proportional to the total traveling cost). Each customer $i \in \mathcal{N}$ must be visited exactly once by one vehicle within a prescribed time interval $[a_i, b_i]$, called a time window, to deliver a quantity q_i of goods. A route starts from the depot and visits a sequence of customers before returning to the depot. It is feasible if the total amount of goods delivered does not exceed the vehicle capacity Q and if it respects the time window of each visited customer.

The VRPTW can be represented on a graph $G = (V, \mathcal{A})$. The node set V contains $|\mathcal{N}| + 2$ nodes: one node for each customer $i \in \mathcal{N}$, as well as one source node o and one sink node d representing the depot at the beginning and the end of the planning horizon, respectively. The arc set $\mathcal{A}$ contains start arcs (o, j), $\forall\, j \in \mathcal{N}$, end arcs (i, d), $\forall\, i \in \mathcal{N}$, and travel arcs (i, j), $\forall\, i, j \in \mathcal{N}$ such that customer j can be visited immediately after customer i in at least one feasible route, that is, if $a_i + t_{ij} \leq b_j$, where t_{ij} is equal to the travel time between i and j plus the service time (if any) at node i. Each arc (i, j) has an associated travel cost (distance) c_{ij}. Note that, in general, c_{ij} is proportional to the travel time or the distance between i and j.

The VRPTW has been well studied in the literature. In the past fifteen years, several exact methods for the VRPTW have been developed. Among them, branch-and-price algo-

rithms (Desrochers *et al.*, 1992; Kohl *et al.*, 1999; Feillet *et al.*, 2004; Irnich et Villeneuve, 2006; Chabrier, 2006; Jepsen *et al.*, 2008; Desaulniers *et al.*, 2008) have produced the best results, mostly because of the quality of the lower bounds yielded by the embedded column generation method. Also, a very large number of heuristics have been proposed (see the survey papers of Bräysy et Gendreau, 2005a,b), including a wide variety of metaheuristics such as tabu search (Cordeau *et al.*, 2001b), evolutionary and genetic algorithms (Berger *et al.*, 2003; Mester et Bräysy, 2005), large neighborhood search (Shaw, 1998; Pisinger et Ropke, 2007), variable neighborhood search (Rousseau *et al.*, 2002; Bräysy, 2003), certain two-phase hybrid approaches (Gehring et Homberger, 2001; Bent et Van Hentenryck, 2004; Homberger et Gehring, 2005), iterated local search algorithms (Ibaraki *et al.*, 2005, 2008), and one parallel cooperative search approach that exploits several known metaheuristics (Le Bouthillier *et al.*, 2005). However, to our knowledge, no heuristics taking advantage of the power of branch-and-price have been proposed in the literature for the VRPTW.

In this paper, we present such a method, namely a large neighborhood search (LNS) algorithm, that relies on a heuristic branch-and-price method for neighborhood exploration. An LNS method is an iterative method where elements of a solution are alternately removed (destruction step) and reinserted (reconstruction step) in order to improve the solution. A neighborhood is thus the set of all solutions containing the subset of elements that have not been removed at a given iteration. Because the size of the neighborhood increases exponentionally with the number of elements removed, it has the potential to change a large portion of the solution, hence its name. Like the approaches of Gehring et Homberger (2001), Bent et Van Hentenryck (2004) and Homberger et Gehring (2005), our method has two phases: in the first, the minimization of the number of vehicles is prioritized; in the second, the priority is changed to reducing total traveled distance with a fixed number of vehicles, namely, the minimum number attained in the first phase. However, as opposed to these three hybrid approaches, our method applies a very similar methodology in both phases.

To evaluate the efficiency of the proposed LNS method, we performed computational experiments on the well-known 56 Solomon's (1987) instances with 100 customers and also on the 300 Gehring et Homberger's (1999) instances involving between 200 and 1000 customers. Compared to the best known methods, our LNS method produced better solutions, especially on the largest instances where the number of vehicles used was significantly reduced. It succeded to compute 145 new best solutions out of the 356 benchmark instances. However, it required more computational time than other leading methods such as that of Pisinger et Ropke (2007).

Combining mathematical programming techniques with metaheuristics is a growing area of research. For instance, De Franceschi *et al.* (2006) developed an LNS algorithm for the

distance-constrained vehicle routing problem (without time windows) where the reconstruction step consists of solving a set partitioning type problem using a commercial mixed-integer programming solver. Their method differs from the one we propose in several ways. Firstly, we use different ad hoc operators to destroy the current solution at each LNS iteration. Secondly, reconstruction is performed using a branch-and-price heuristic. Finally, the first phase of our two-phase method aims at minimizing the number of vehicles used, while De Franceschi *et al.* consider a fixed number of vehicles. Our work thus brings an interesting contribution in this research trend.

The paper is organized as follows. In the next section, we present the LNS algorithm framework, with an emphasis on the description of the two-phase solution process. In Section 4.3, we present the set of destruction operators that can be used to define the neighborhood to explore at each iteration of the LNS algorithm. Most of these operators are adaptations of operators already proposed in the literature. Section 4.4 describes the branch-and-price heuristic applied in the reconstruction step at each LNS iteration. This heuristic is composed of a heuristic branching strategy and a heuristic column generation method where columns (routes) are generated by tabu search. The results of our computational experiments are reported in Section 4.5. Finally, conclusions are drawn in Section 4.6.

4.2 Algorithm framework

An LNS algorithm is an iterative process that destroys at each iteration a part of the current solution using a chosen neighborhood definition procedure and reconstructs it in the hope of finding a better solution. Figure 4.1 illustrates the framework of our LNS algorithm that works in a two-phase fashion. The first one, the vehicle number reduction phase (VNR), as its name indicates, tries to reduce the total number of vehicles used in the current solution, while the other one, the total distance reduction phase (TDR) tries to reduce the total mileage for a fixed number of vehicles. In this figure, the numbers in circles indicate the algorithm step numbers. Note that the two main steps (destruction and reconstruction) of an LNS algorithm appear in double-lined rectangles (Steps 8 and 9). The details of these two steps will be given in the following sections.

The algorithm starts in Step 1 by computing an initial solution using Solomon's (1987) I1 insertion heuristic. In Step 2, it computes a lower bound m_{lb} on the optimal number of vehicles using the vehicle capacity Q and the total customer demand:

$$m_{lb} = \left\lceil \frac{\sum_{i \in \mathcal{N}} q_i}{Q} \right\rceil . \tag{4.1}$$

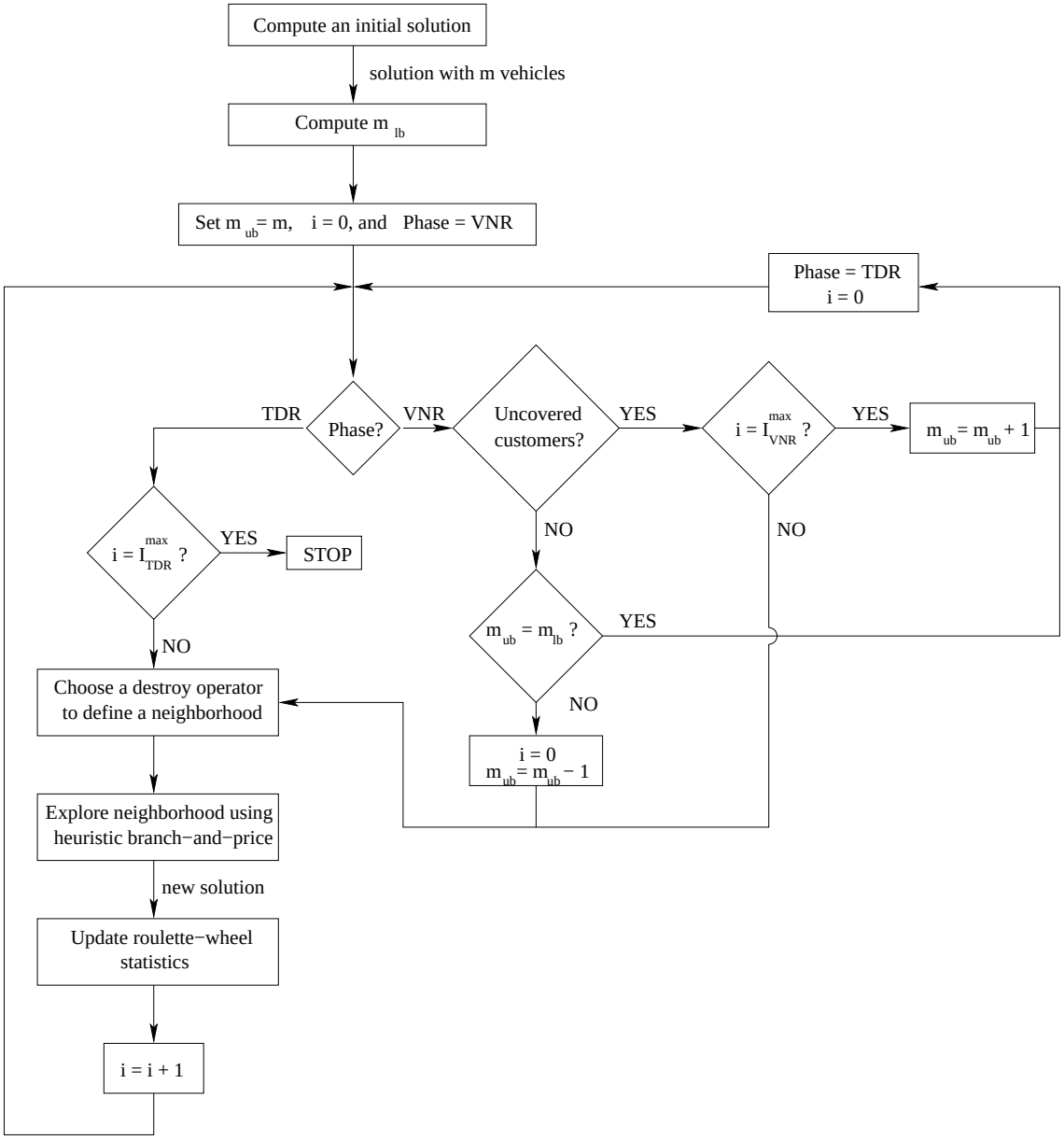

Figure 4.1 Algorithm framework

It also sets an upper bound m_{ub} on this number to the number of vehicles used in the initial solution, an iteration counter i to 0, and the current phase at VNR (Step 3).

The VNR phase is skipped if $m_{ub} = m_{lb}$ (Step 6). Otherwise, it begins by lowering the upper bound m_{ub} by one (Step 7). This upper bound is enforced at each iteration during reconstruction, while allowing (with a penalty cost) some customers not to be serviced. If no feasible solution (covering all customers) can be found within a prespecified number of iterations I_{VNR}^{max} (Step 12), the search is abandoned for that bound and the TDR phase is started from the best feasible solution found so far. Otherwise, the upper bound m_{ub} is lowered again by one (Step 7) and the algorithm has another I_{VNR}^{max} iterations to find a feasible solution and so on. Note that, whenever a feasible solution with m_{lb} vehicles is obtained (case yes in Step 6), the VNR phase is stopped and the TDR phase starts from that last solution. In the TDR phase, I_{TDR}^{max} iterations are performed (tested in Step 15).

At each iteration of the LNS algorithm (either in the VNR or the TDR phase), a neighborhood-defining operator is selected (Step 8) within a set of four different operators using a roulette-wheel that favors operators which were the most successful at improving the current solution in the past iterations. These four operators are described in Sections 4.3.1 to 4.3.4, while the roulette-wheel procedure is discussed in Section 4.3.5. The selected operator is then applied to destroy parts of the current solution. The destruction consists of determining a subset of customers (hereafter called the removed customers) which are disconnected from their current routes, leaving partial routes and isolated customers. The neighborhood contains all solutions that respect the partial routes. These solutions must be feasible except, during the VNR phase, where they might not cover all customers. Reconstruction is performed afterwards in Step 9 by solving the resulting reoptimization problem that corresponds to the original VRPTW where parts of the current solution routes are fixed. This restricted VRPTW is solved using a branch-and-price heuristic to create a new solution. This heuristic is described in Section 4.4.1. Roulette-wheel statistics are then updated (Step 10) and the process starts over again.

4.3 Destruction

To diversify the search, we consider four different neighborhood operators that can be used to destroy the current solution in Step 8 of the algorithm. Each operator tries to select a predetermined number of customers that are to be removed from the current solution. In the VNR phase, uncovered customers might not to be selected except when there are less than ten of them, in which case they are all removed from the solution. When a covered customer is removed, adjacent arcs are removed from the solution, leaving partial routes.

To choose at each iteration which operator to use, a roulette-wheel procedure is invoked. Each operator and the roulette-wheel are slightly modified in the VNR phase to promote the insertion of uncovered customers into routes. Note that the part of the solution that has not been destroyed is fixed in the reconstruction process, including the uncovered customers that have not been selected (they will remain uncovered).

4.3.1 Proximity operator

The proximity operator is an extension of an operator proposed by Shaw (1998). Its goal is to select customers that are related geographically and temporally. An initial customer is chosen randomly and added to an empty pool of removed customers. For each additional customer, the selection procedure has three steps as follows. First, a seed customer i is chosen randomly in the pool of removed customers. Second, all remaining customers are ranked in decreasing order of a spatio-temporal proximity measure to i. This measure is defined by

$$R(i,j) = \frac{1}{(\frac{min\{c_{ij},c_{ji}\}}{c_i^{max}} + \frac{1}{T_{ij}+T_{ji}})}, \tag{4.2}$$

where c_{ij} is the cost of the arc from i to j and c_i^{max} is the largest arc cost $c_{i\ell}$ or $c_{\ell i}$ for all customers ℓ that have not been removed yet. Similar to a time window proximity measure defined by Gendreau *et al.* (1995), T_{ij} is equal to $max\{1, min\{b_j, b_i+t_{ij}\}-max\{a_j, a_i+t_{ij}\}\}$. In this expression, $min\{b_j, b_i+t_{ij}\}-max\{a_j, a_i+t_{ij}\}$ is the width of the interval of the feasible visiting times at customer j when customer i is visited immediately before j, if possible. If an arc exists from i to j but not from j to i, we set $c_{ji} = \infty$, and, if no arc exists either way, $R(i,j)$ takes value 1. In the third step, the next customer to remove is chosen randomly among the remaining customers, favoring those having a greater proximity measure to i. Indeed, the rank of the customer to be chosen is $\lceil N\rho^D \rceil$, where N is the number of remaining customers, ρ a number generated randomly between 0 and 1, and D a constant greater or equal to 1. A D value of 1 results in complete randomness while an infinite value yields complete determinism. For our experiments, the value of D was set at 35.

In the VNR phase, the seed customer is chosen randomly among the uncovered customers. The other uncovered customers are removed only if they are chosen according to their spatio-temporal proximity.

4.3.2 Route portion operator

The proximity operator described above can remove single customers on certain routes, leaving almost no flexibility to change these routes. To avoid this drawback, the route portion

operator, presented as the SMART (SMAll RouTing) operator by Rousseau *et al.* (2002), removes portions of routes around pivot customers. The first pivot is chosen randomly, then adjacent customers on the same route of the current solution are removed as well. A second pivot customer on another route is chosen based on the spatio-temporal proximity measure to the first pivot (defined in the preceding section). Adjacent customers are removed and a third pivot is chosen based on the proximity measure to the first pivot as well, and so on until enough customers are removed or all routes are covered by a pivot since there can only be one pivot per route.

Adjacent customers are chosen according to a maximum distance to the pivot customer (distance is measured along the route, not directly between the customers). All customers on each side of the pivot within the maximum distance as well as the first customer outside of this distance are removed. The maximum distance $\bar{d}$ is set once per call to this operator as follows. Let j be the first pivot customer, and i and k its immediate preceding and succeding customers (or depot) on its route, respectively. Then, $\bar{d} = f_{rp} \max\{c_{ij}, c_{jk}\}$, where the multiplicator f_{rp} evolves throughout the solution process. The first time the route portion operator is called, f_{rp} has a value of 1. If less than the target number of customers have been chosen after selecting a pivot in each route, f_{rp} is multiplied by this target number divided by the number of actually removed customers for the next call to the operator. This ensures that, after a few calls to this operator, the right number of customers can be removed from solutions having fewer routes. Furthermore, f_{rp} also adapts to the fact that the distance between adjacent customers typically lowers as the solution process evolves.

Like for the proximity operator, the first pivot in the VNR phase is chosen randomly between the uncovered customers and, in this case, the maximum distance is equal to the distance from this pivot to the depot. Other uncovered customers can be chosen as subsequent pivots. When an uncovered customer is selected as a pivot, no adjacent customers are removed.

4.3.3 Longest detour operator

The longest detour operator was presented as well by Rousseau *et al.* (2002). The purpose of this operator is to remove the customers yielding in the current solution the largest distance increases for servicing them. All the customers are first ordered decreasingly by the detour they generate in the current solution. For a customer j being serviced between i and k (customers or depot), its associated detour is equal to $c_{ij} + c_{jk} - c_{ik}$. Like for the proximity operator, the customers are removed randomly, favoring those generating a greater detour.

In the VNR phase, uncovered customers do not generate any detour. Therefore, to also select uncovered customers in this phase, each time that a covered customer i is removed

according to its detour, the uncovered customer which is the closest to customer i according to the spatio-temporal proximity measure is also removed unless it was previously removed. In this latter case, no other uncovered customer is selected with respect to i.

4.3.4 Time operator

The time operator simply removes customers that are serviced almost at the same time. It first selects randomly a specific time t_s within the horizon. The customers are then ordered increasingly according to a value v_i defined for each customer i as

$$v_i = \begin{cases} a'_i - t_s & \text{if } t_s < a'_i \\ 0 & \text{if } a'_i \leq t_s \leq b'_i \\ t_s - b'_i & \text{if } t_s > b'_i \end{cases} \tag{4.3}$$

where $[a'_i, b'_i]$ is the interval of times at which customer i can be visited along its route in the current solution while it remains feasible. The bounds of this interval are computed recursively along this route as follows:

$$a'_{i(0)} = a_{min} \tag{4.4}$$

$$a'_{i(r)} = max\{a_{i(r)}, a'_{i(r-1)} + t_{i(r-1),i(r)}\} \qquad r = 1, \ldots, r_{max} \tag{4.5}$$

$$b'_{i(r_{max})} = b_{max} \tag{4.6}$$

$$b'_{i(r)} = min\{b_{i(r)}, b'_{i(r+1)} - t_{i(r),i(r+1)}\} \qquad r = r_{max} - 1, \ldots, 0 \tag{4.7}$$

where r is the rank of the customer on the route, ranks 0 and r_{max} correspond both to the depot, and $i(r)$ is the customer (or depot) in rank r. a_{min} and b_{max} are the earliest departure time from the depot and the latest arrival time at the depot, respectively. Once the customers are ordered, they are chosen with the same procedure presented for the proximity operator.

In the VNR phase, the time interval of an uncovered customer i is its time window $[a_i, b_i]$.

4.3.5 Roulette-wheel procedure

At each iteration of the LNS algorithm, the roulette-wheel procedure selects in Step 8 which operator is applied to define the neighborhood. It is similar to the procedure introduced in Pisinger et Ropke (2007). Each destruction operator is assigned a value π_i that starts at 5. Each time that an operator i is selected and allows to improve the current solution, π_i is incremented by 1. At the beginning of each LNS iteration, the destruction operator is selected randomly, where each operator i has a probability of $\frac{\pi_i}{\sum_i \pi_i}$ of being selected.

In the VNR phase, the π_i value of an operator that yields a solution with fewer uncovered customers is increased by 2 in order to prioritize operators which are best at reducing the number of uncovered customers.

4.4 Reconstruction

At each iteration of the LNS algorithm, the VRPTW restricted to the selected neighborhood can be modeled as a set partitioning problem where the variables are feasible routes. Let Ω be a subset of all feasible routes (respecting the fixed parts of the solution). With each route $p \in \Omega$, associate the following parameters: c_p, its cost and $v_{ip}, \forall i \in \mathcal{N}$, taking value 1 if customer i is serviced by route p and 0 otherwise. Finally, a binary variable θ_p is defined for each route $p \in \Omega$, taking a value of 1 if route p is part of the solution and 0 otherwise. With this notation, the restricted VRPTW can be formulated as:

$$\text{Minimize} \qquad \sum_{p \in \Omega} c_p \theta_p \tag{4.8}$$

$$\text{subject to:} \qquad \sum_{p \in \Omega} v_{ip} \theta_p = 1, \qquad \forall\, i \in \mathcal{N} \tag{4.9}$$

$$\sum_{p \in \Omega} \theta_p \leq m_{ub}, \tag{4.10}$$

$$\theta_p \text{ binary}, \qquad \forall\, p \in \Omega. \tag{4.11}$$

The objective function (4.8) aims at minimizing total cost. Set partitioning constraints (4.9) ensure that each customer is visited exactly once by one vehicle. Constraint (4.10) imposes an upper bound on the number of vehicles that can be used. Finally, (4.11) provide binary requirements on the variables.

In general, an LNS algorithm must perform many iterations to reach very good quality solutions. Therefore, to keep computational times relatively low, the reconstruction process must be fast and effective. We thus propose to solve, in Step 9 of the overall algorithm, model (4.8)–(4.11) using a heuristic branch-and-price method, that is, a heuristic column generation method embedded into a heuristic branch-and-bound search. The column generation method and the branching scheme are described in the following sections.

4.4.1 Heuristic column generation

Column generation is used to solve the linear relaxation of model (4.8)-(4.11). Column generation is an iterative process that solves at each iteration this linear relaxation restricted to a subset of the variables. This restricted problem is called the restricted master problem (RMP). The dual solution of the RMP is then transfered to a subproblem whose objective is to generate negative reduced cost variables to be added back to the RMP. The latter is then solved again with the augmented subset of variables. An exact column generation method ends when the optimal solution of the subproblem has a nonnegative reduced cost. We can conclude thereof that the solution to the RMP is optimal for the whole linear relaxation because no more negative reduced cost variables exist.

The RMP is solved by a linear programming algorithm such as the primal simplex method. For the VRPTW, the column generation subproblem is an elementary shortest path problem with resource constraints (ESPPRC) defined on a restricted network which guarantees that the fixed parts of the routes in the current solution remain untouched. The ESPPRC is NP-hard (Dror, 1994) but can be solved heuristically. The proposed column generation heuristic is an adaptation of the exact method developed by Desaulniers *et al.* (2008). In their method, a sequence of column generators with varying solution times is used at each iteration to generate negative reduced cost variables. At each iteration, the first generator invoked is a tabu search algorithm that rapidly finds negative reduced cost columns in most iterations. When failing to do so, an attempt is made to generate such columns using a heuristic dynamic programming algorithm. If failure occurs again, an exact dynamic programming algorithm, is called to ensure the optimality of the solution process. In the proposed heuristic, both dynamic programming procedures are omitted and the tabu search method is used as the sole column generator. The subproblem and the tabu search method used to solve it are described in Sections 4.4.1 and 4.4.1, respectively.

Two strategies are used to speed up the reconstruction process. First, for large instances (400 customers or more), column generation is prematurely halted when no improvement in the objective value of the RMP has been realized in the last I_{CG}^{max} column generation iterations (I_{CG}^{max} was set at 5 for our experiments). Second, the column generation heuristic is warm started at each LNS iteration by introducing, into the first RMP, variables that were generated in previous LNS iterations and whose corresponding routes are still valid for the current neighborhood. The management of these variables is explained in Section 4.4.1.

Subproblem

As mentioned above, the subproblem is an ESPPRC which can be defined over the network $G = (V, \mathcal{A})$ described in the introduction. However, to compute the path with the least reduced cost when solving the subproblem, the costs c_{ij}, $(i, j) \in \mathcal{A}$, are replaced by reduced costs

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \alpha_i & \text{if } i \in \mathcal{N} \\ c_{ij} - \mu & \text{if } i = o, \end{cases}$$

where α_i, $i \in \mathcal{N}$, and μ are the dual variable values of the RMP constraints (4.9) and (4.10) at the current column generation iteration, respectively.

Each feasible vehicle route can be represented by a path in G. However, resource constraints (see Irnich et Desaulniers, 2005) are required in the subproblem to ensure the feasibility of the path with respect to time windows, vehicle capacity, and elementarity. A resource is a quantity that accumulates along a path and is restricted to an interval at each node. For further details about the ESPPRC subproblem, consult Irnich et Desaulniers (2005).

Tabu search

The tabu search method used to solve the ESPPRC was proposed by Desaulniers *et al.* (2008) to quickly generate negative reduced cost columns. Tabu search (see Glover et Laguna, 1997) is a metaheuristic that has been successful at solving a wide variety of combinatorial optimization problems. It is an iterative method that starts from an initial solution and applies moves to improve it. Possible moves can be defined by a set of operators and are generally quite simple. A neighbor is a solution that differs from a current solution by only one move, and at each iteration, the move creating the best neighbor is chosen even if the objective value is deteriorated. To avoid cycling, a memory of past moves, the tabu list, is kept in order to forbid recent moves to be reversed for a number of iterations. This allows the search to escape from local minima.

For the ESPPRC, the tabu method that we use relies on two operators, inserting a customer in the current route and removing a customer from this route. Each time that a customer is inserted or removed, the reverse move becomes tabu for a fixed number of iterations. Therefore, at each iteration, all possible moves are to remove every (non-tabu) customer individually from the route and to insert every other (non-tabu) customer individually at every possible insertion point in the route. The search space is limited to only feasible routes. Thus, for each insertion move, route feasibility is checked in terms of time windows and vehicle capacity. No feasibility checks are needed for customer removal moves. Since sequences of customers can be fixed with respect to the neighborhood defined by the destruction operators,

the tabu method treats these sequences as aggregated customers that cannot be visited separately.

To diversify the search, the tabu search method is started multiple times from a set of different initial solutions and limited to a maximum number $I_{max}^{tabu,sol}$ to be performed for each initial solution. The set of initial solutions differs in both phases of our algorithm. In the VNR phase, only the routes associated with non-degenerate variables in the current RMP solution are used. Because the number of these variables can vary, the total number of tabu search iterations per column generation iteration $I_{max}^{tabu,tot}$ is fixed and evenly divided among these variables. In the TDR phase, the routes associated with all basic variables (regardless of their value) are used as initial solutions. In both cases, all initial solutions have a reduced cost of 0, and are thus very good initial solutions since we are looking for negative reduced cost values.

Long-term column memory

At the end of each iteration of the LNS algorithm, the columns present in the last RMP solved are added to a pool of columns that can be reused in subsequent iterations. After defining the neighborhood at each iteration, the pool of columns is scanned to find columns that are valid with respect to the neighborhood structure. These columns are added to the RMP to warm start it. This procedure typically reduces the number of column generation iterations and gives access to columns that tabu search may not be able to generate. With these initial columns, column generation is usually faster and can produce higher quality heuristic solutions. However, managing this pool of columns requires a significant amount of computational time at each LNS iteration. This management time often outweighs the time saved by the column generation heuristic when the pool contains a very large number of columns. Therefore, we limit the number of columns in this pool to benefit from them while tempering the resulting pool management time. When the number of columns in memory is greater than a predetermined upper limit at the end of an LNS iteration, columns are eliminated randomly from the pool until a lower limit is reached (set at 70% of the upper limit).

4.4.2 Branching strategy

In order to quickly derive integer solutions, we propose to use an effective heuristic branching method. As suggested in Desaulniers *et al.* (2002), we impose decisions on the route variables θ_p and explore the search tree using a depth-first strategy without backtracking. At the end of each linear relaxation, when a fractional solution is found, the route variable with

the largest fractional value is simply fixed at 1. No backtracking is allowed, that is, branching decisions cannot be reversed to go up in the search tree. Because of this, the solution found at the end of the branching process might be worst than the previous one or even no feasible solution might be found. In the former case, the deteriorated solution is kept to contribute to the diversification of the search. In the latter case, the previous solution is reused as the current solution.

4.5 Computational experiments

We tested our method on the well-known benchmark VRPTW instances of Solomon (1987) and Gehring et Homberger (1999). All our tests were performed on an AMD OPTERON processor clocked at 2.3 GHz. The branch-and-price heuristic was implemented using the Gencol software library, version 4.5, (developed at the GERAD research center in Montreal) and relied on the CPLEX solver, version 9.1, for solving the RMP. This section describes the instances, gives the parameter setting, and reports the main results before providing the results of a sensitivity analysis on certain parameter values.

4.5.1 Instances

Solomon (1987) designed 56 VRPTW instances with 100 customers divided into six classes: R1, R2, RC1, RC2, C1 and C2, each containing between 8 and 12 instances. In the R class instances, the customers are distributed randomly in the space. In the C class instances, they are clustered. The RC class instances have a mixed distribution of customers. Type 1 instances have a shorter scheduling horizon and thus allow fewer customers to be serviced per route, while type 2 instances have a longer scheduling horizon. The average width of the time windows and the number of customers with constraining time windows also vary from one instance to another within the same class. Based on the same principles, Gehring et Homberger (1999) introduced larger VRPTW instances involving 200, 400, 600, 800, and 1000 customers, with 10 instances in each class for each size.

In the following, instances are identified with ids. Each instance id starts with its class, followed by a number indicating its size, and another number corresponding to the rank of the instance within its class. For example, instance R2_6_9 is the ninth instance of the class R2 with 600 customers.

4.5.2 Parameter values

All parameter values have been adjusted through a series of preliminary tests on a subset of the instances. Table 4.1 provides the parameter values that were not given previously and

that were used for the results presented in Subsection 4.5.3. These parameters are the most sensible ones and will be subject to a sensitivity analysis in Subsection 4.5.4. Note that for the number of customers removed at each LNS iteration, we used two different values: 70 for the small instances (with 100 and 200 customers) and 100 for the larger ones.

4.5.3 Main results

Tables 4.2 through 4.7 present the results of our algorithm on the benchmark instances and those of the best algorithms that can be found in the literature. For the 100-customer instances (Table 4.2), these are the algorithms of Pisinger et Ropke (2007), Bent et Van Hentenryck (2004), Bräysy (2003) and Ibaraki *et al.* (2005) which are abbreviated by RP, BVH, B, and IIKMUY, respectively. For the larger instances (Tables 4.3–4.7), the algorithms of Pisinger et Ropke (2007), Gehring et Homberger (2001), Mester et Bräysy (2005), Le Bouthillier *et al.* (2005), abbreviated by RP, GH, MB, and LCK, respectively, serve as a comparison basis. Note that the algorithm of Homberger et Gehring (2005) has not been retained because the best results they report were obtained using multiple runs with multiple parameter configurations for each instance. Furthermore, the results of Ibaraki *et al.* (2008) are also not considered because they were derived using the minimum number of vehicles reported in the literature for each instance.

For each instance class and each method, we provide in the following tables two values: the mean number of vehicles used and the mean total traveled distance. Rows *CNV* and *CTD* show, respectively, the cumulative number of vehicles and the cumulative total distance for all instances in the data set. Row *CPU* presents the type of processor used and row *Time (min)*, the average time in minutes taken by the algorithm for solving an instance once. The last row, *Runs*, gives the number of runs performed in order to obtain the results. The results of our algorithm (PDR) are, however, presented in two columns. The first one, *Best*, indicates the best results obtained in five runs, while the second one, *Avg*, specifies the average value of the solutions obtained. Bold numbers highlight the best results for each class.

On these main runs, our algorithm succeded to improve the best known solution value of 106 instances (out of 356), compared to values reported on the Sintef website [1], as of August 30, 2007, and the website [2] presenting the detailed results of Ibaraki *et al.* (2008). In all the experiments performed during this project including parameter tuning and sensitivity analysis, a total of 145 new best solutions were found. The values of these new solutions are reported in Table 4.8. Bold values indicate solutions reducing the number of vehicles compared to the previously best known solutions.

1. www.sintef.no/static/am/opti/projects/top/vrp/benchmarks.html
2. www.al.cm.is.nagoya-u.ac.jp/~yagiura/papers/vrpctw_abst.html

Parameter	Instance sizes	Value
Number of customers removed	100-200	70
	400-1000	100
Maximum number of columns in memory pool	100-1000	50000
Number of tabu iterations per column generation iteration in VNR phase ($I_{max}^{tabu,tot}$)	100-1000	$\|\mathcal{N}\|$
Number of tabu iterations per initial solution in TDR phase ($I_{max}^{tabu,sol}$)	100-1000	5
Maximum number of VNR iterations to find a feasible solution (I_{max}^{VNR})	100-1000	600
Number of TDR iterations (I_{max}^{TDR})	100-1000	800

Table 4.1 Parameter values

Compared to other methods, our LNS algorithm finds better CNVs and CTDs for all instance sizes, when taking the best of five runs (column *Best*). For instances with 600 customers or less, we obtained solutions that exhibit the same CNVs as the best known CNVs, but lower CTDs. For the larger instances, the CNVs were significantly reduced (from 2758 to 2745 and from 3438 to 3432 for the 800- and 1000-customer instances, respectively). Note, however, that when we obtain the best cumulative results for a given size, we do not obtain the best results for all instance classes. On average (see the *Avg* columns), the quality of the solutions produced by our algorithm remains very good, especially for the instances with 400 customers or more. Indeed, the average CNVs and CTDs are next to the best for the 400- and 600-customer instances, and the best for the 800- and 1000-customer instances. These results show that our method is very efficient and robust at finding very good quality solutions for the VRPTW. It is however somewhat slow when compared to other methods like that of Pisinger et Ropke (2007).

In order to verify the efficiency of the destruction operators, statistics on the performance of the operators were gathered from the main runs. Table 4.9 provides two values for each operator in each phase for the data sets involving between 400 and 1000 customers. The first one, *nb calls*, is the total number of LNS iterations in the corresponding phase in which the given operator was called as the destruction operator, and the second one, *nb impr*, is the total number of times among these iterations that the current solution was improved. The row *Total* gives the total numbers of calls and improvements over all datasets, while the last row indicates the percentage of times that each operator improves the current solution when called. From these results, we observe that the behavior of each operator is similar from one instance size to another and that the percentage of improved solutions found is smaller in

	PDR		PR	BVH	B	IIKMUY
	Best	Avg	(2007)	(2004)	(2003)	(2005)
R1	**11.92**	11.92	11.92	11.92	11.92	11.92
	1210.34	1211.69	1212.39	1211.10	1222.12	1217.40
R2	2.73	2.85	2.73	**2.73**	2.73	2.73
	955.74	939.861	957.72	**954.27**	975.12	959.11
RC1	**11.50**	11.50	11.50	11.50	11.50	11.50
	1384.16	1386.98	1385.78	1384.17	1389.58	1391.03
RC2	**3.25**	3.28	3.25	3.25	3.25	3.25
	1119.44	1115.68	1123.49	1124.46	1128.38	1122.79
C1	**10.00**	**10.00**	**10.00**	**10.00**	**10.00**	**10.00**
	828.38	**828.38**	**828.38**	**828.38**	**828.38**	**828.38**
C2	**3.00**	**3.00**	**3.00**	**3.00**	**3.00**	**3.00**
	589.86	**589.86**	**589.86**	**589.86**	**589.86**	**589.86**
CNV	**405**	406.6	405	405	405	405
CTD	**57240**	57101	57332	57273	57710	57444
CPU	OPT 2.3Ghz	OPT 2.3Ghz	P4 3Ghz	SU 10	P-200Mhz	P3 1Ghz
Time (min)		30	2.5	120	82.5	250
Runs		5	10	>5	1	1

Table 4.2 Solomon's instances with 100 customers

	PDR		PR	GH	MB	LCK
	Best	Avg	(2007)	(2001)	(2005)	(2005)
R1	18.2	18.20	18.2	18.2	18.2	**18.2**
	3615.69	3624.10	3631.226	3885.03	3618.68	**3615.06**
R2	**4.0**	4.02	4.0	4.0	4.0	4.0
	2937.67	2949.28	2949.368	3032.49	2942.92	2969.90
RC1	**18.0**	18.00	18.0	18.1	18.0	18.0
	3192.56	3211.43	3212.282	3674.91	3221.34	3255.97
RC2	4.3	4.38	**4.3**	4.4	4.4	4.3
	2559.32	2540.23	**2556.874**	2671.34	2519.79	2584.18
C1	18.9	18.90	18.9	18.9	**18.8**	18.9
	2718.77	2721.71	2721.522	2842.08	**2717.21**	2736.84
C2	**6.0**	6.00	6.0	6.0	6.0	6.0
	1831.59	1831.88	1832.947	1856.99	1833.57	1833.91
CNV	**694**	695	694	696	694	694
CTD	**168556**	168786	169042	179328	168573	169958
CPU	OPT 2.3Ghz	OPT 2.3Ghz	P4 3Ghz	P-400Mhz	P4 2Ghz	P3 850MHz
Time (min)		53	7.7	4x2.1	8	5x10
Runs		5	10	3	1	1

Table 4.3 Gehring and Homberger's instances with 200 customers

	PDR		PR	GH	MB	LCK
	Best	Avg	(2007)	(2001)	(2005)	(2005)
R1	36.4	36.40	36.4	36.4	**36.3**	36.4
	8420.52	8451.44	8540.04	9478.22	**8530.03**	8607.97
R2	8.0	8.00	8.0	8.0	**8.0**	8.0
	6213.48	6278.74	6241.72	6650.28	**6209.94**	6302.08
RC1	**36.0**	36.00	36.0	36.1	36.0	36.0
	7940.65	8002.87	8069.30	9294.99	8066.44	8267.81
RC2	8.6	8.80	**8.5**	8.8	8.8	8.6
	5269.09	5290.13	**5335.09**	5629.43	5243.06	5397.54
C1	**37.6**	37.62	37.6	38.0	37.9	37.9
	7182.75	7199.78	7290.16	7855.82	7148.27	7223.06
C2	**11.9**	11.98	12.0	12.0	12.0	12.0
	3874.58	3854.17	3844.69	3940.19	3840.85	3862.66
CNV	**1385**	1388.0	1385	1392	1389	1389
CTD	**389011**	390771	393210	428489	390386	396611
CPU	OPT 2.3Ghz	OPT 2.3Ghz	P4 3Ghz	P-400Mhz	P4 2Ghz	P3 850MHz
Time (min)		89	15.8	4x7.1	17	5x20
Runs		5	10	3	1	1

Table 4.4 Gehring and Homberger's instances with 400 customers

	PDR		PR	GH	MB	LCK
	Best	Avg	(2007)	(2001)	(2005)	(2005)
R1	**54.5**	54.50	54.5	54.5	54.5	54.8
	18252.13	18359.92	18888.52	21864.47	18358.68	18698.37
R2	11.0	11.00	**11.0**	11.0	11.0	11.2
	12808.59	12974.58	**12619.26**	13656.15	12703.52	12989.35
RC1	**55.0**	55.00	55.0	55.0	55.0	55.2
	16266.14	16376.34	16594.94	19114.02	16418.63	16643.27
RC2	11.7	11.94	**11.6**	11.9	12.1	11.8
	10990.85	10926.01	**10777.12**	11670.29	10677.46	10868.94
C1	**57.4**	57.40	57.5	57.7	57.8	57.7
	14106.03	14134.81	14065.89	14817.25	14003.09	14166.80
C2	**17.5**	17.60	17.5	17.8	17.8	17.9
	7632.37	7646.82	7801.296	7889.96	7455.83	7582.61
CNV	**2071**	2074.4	2071	2079	2082	2086
CTD	**800797**	805325	807470	890121	796172	809493
CPU	OPT 2.3Ghz	OPT 2.3Ghz	P4 3Ghz	P-400Mhz	P4 2Ghz	P3 850MHz
Time (min)		105	18.3	4x12.9	40	5x30
Runs		5	5	3	1	1

Table 4.5 Gehring and Homberger's instances with 600 customers

	PDR		PR	GH	MB	LCK
	Best	Avg	(2007)	(2001)	(2005)	(2005)
R1	**72.8**	72.80	72.8	72.8	72.8	72.8
	31797.42	31949.31	32316.79	34653.88	31918.47	32290.48
R2	15.0	15.00	15.0	15.0	**15.0**	15.0
	20651.81	20845.50	20353.51	21672.85	**20295.28**	20765.88
RC1	**72.0**	72.00	73.0	72.3	73.0	72.3
	33170.01	33756.19	29478.3	40532.35	30731.07	37075.19
RC2	15.8	16.12	**15.7**	16.1	15.8	15.8
	16852.38	16927.65	**16761.95**	17941.23	16729.18	17202.08
C1	**75.4**	75.54	75.6	76.1	76.2	76.2
	25093.38	25097.40	25193.13	26936.68	25132.27	25612.47
C2	**23.5**	23.60	23.7	23.7	23.7	24.0
	11569.39	11578.86	11725.46	11847.92	11352.29	11393.80
CNV	**2745**	2750.6	2758	2760	2765	2761
CTD	**1391344**	1401549	1358291	1535849	1361586	1443399
CPU	OPT 2.3Ghz	OPT 2.3Ghz	P4 3Ghz	P-400Mhz	P4 2Ghz	P3 850MHz
Time (min)		129	22.7	4x30.1	145	5x40
Runs		5	5	3	1	1

Table 4.6 Gehring and Homberger's instances with 800 customers

	PDR		PR	GH	MB	LCK
	Best	Avg	(2007)	(2001)	(2005)	(2005)
R1	**91.9**	91.90	92.2	91.9	92.1	92.0
	49702.32	50168.00	50751.25	58069.61	49281.48	51847.22
R2	19.0	19.00	**19.0**	19.0	19.0	19.0
	30495.26	30730.35	**29780.82**	31873.62	29860.32	30441.05
RC1	90.0	90.00	90.0	90.1	**90.0**	90.0
	45574.11	45924.74	46752.15	50950.14	**45396.41**	46118.08
RC2	18.5	18.82	**18.3**	18.5	18.7	18.5
	25470.33	25464.40	**25090.88**	27175.98	25063.51	25390.40
C1	**94.3**	94.50	94.6	95.4	95.1	95.1
	41783.27	41913.42	41877.00	43392.59	41569.67	42403.21
C2	**29.5**	29.56	29.7	29.7	29.7	29.6
	16657.06	16817.76	16840.37	17572.72	16639.54	17164.51
CNV	**3432**	3437.8	3438	3446	3446	3442
CTD	**2096823**	2110187	2110925	2290367	2078110	2133645
CPU	OPT 2.3Ghz	OPT 2.3Ghz	P4 3Ghz	P-400Mhz	P4 2Ghz	P3 850MHz
Time (min)		162	26.6	4x30.1	600	5x50
Runs		5	5	3	1	1

Table 4.7 Gehring and Homberger's instances with 1000 customers

	R1		R2		RC1		RC2		C1		C2	
#	Veh	Dist	Veh	Dist	Veh	Dist	Veh	Dist	Veh	Dist	Veh	Dist
200 customers												
1	-	-	4	4483.16	18	3602.80	6	3099.53	-	-	-	-
2	18	4040.60	4	3621.20	18	3249.50	5	2825.24	-	-	-	-
3	-	-	-	-	18	3008.76	4	2603.83	18	2707.35	-	-
4	18	3059.22	-	-	-	-	-	-	18	2643.97	6	1703.43
5	18	4107.86	4	3366.79	18	3385.88	4	2911.46	-	-	-	-
6	18	3583.14	4	2913.03	18	3324.80	-	-	-	-	-	-
7	18	3150.11	-	-	18	3189.32	4	2526.18	-	-	-	-
8	18	2952.65	-	-	18	3083.93	4	2293.35	-	-	-	-
9	18	3760.58	4	3092.53	-	-	-	-	-	-	-	-
10	18	3302.72	-	-	18	3008.53	-	-	-	-	-	-
400 customers												
1	-	-	8	9213.68	36	8630.94	11	6688.31	-	-	-	-
2	36	8955.50	8	7641.67	36	7958.67	-	-	36	7687.38	-	-
3	36	7841.52	-	-	36	7562.60	8	4958.74	36	7060.73	**11**	**4109.88**
4	36	7318.62	8	4297.20	36	7332.59	-	-	-	-	**11**	**3865.45**
5	36	9242.43	-	-	36	8249.63	9	5923.95	-	-	-	-
6	36	8383.67	-	-	36	8223.12	8	5863.56	-	-	-	-
7	36	7656.94	-	-	36	8001.12	8	5466.70	39	7417.92	-	-
8	36	7293.69	-	-	36	7836.29	8	4848.87	37	7363.51	-	-
9	36	8750.84	-	-	36	7811.55	-	-	36	7061.21	12	3865.65
10	36	8125.03	-	-	36	7668.77	8	4311.59	36	6860.63	**11**	**4115.46**
600 customers												
1	-	-	11	18291.18	55	17317.13	-	-	-	-	-	-
2	54	19147.38	-	-	55	16123.40	-	-	56	14163.31	17	8380.49
3	54	17216.16	-	-	55	15358.13	-	-	56	13781.19	17	7595.43
4	54	15947.03	-	-	-	-	-	-	-	-	-	-
5	54	20017.80	-	-	55	16934.45	12	12168.79	-	-	-	-
6	54	18237.76	-	-	55	16842.27	-	-	-	-	18	7472.24
7	54	16796.63	-	-	55	16450.42	-	-	58	14851.65	18	7512.33
8	-	-	-	-	55	16164.82	-	-	**56**	**14541.53**	17	7778.30
9	54	19015.51	-	-	-	-	-	-	56	13718.23	-	-
10	54	18204.18	-	-	55	15936.81	-	-	56	13669.88	-	-
800 customers												
1	-	-	15	28392.87	**72**	**35102.79**	19	20520.49	-	-	-	-
2	72	33190.68	-	-	72	33361.67	-	-	74	25528.55	**23**	**12332.37**
3	72	29943.87	-	-	72	30608.16	-	-	72	24366.83	23	11438.72
4	-	-	-	-	72	28363.65	-	-	-	-	-	-
5	72	34247.99	-	-	**72**	**34481.02**	16	18917.65	-	-	-	-
6	72	31728.99	-	-	**72**	**34849.96**	15	18600.22	-	-	-	-
7	72	29399.21	-	-	72	33102.75	-	-	**77**	**26639.13**	24	11380.54
8	72	28191.89	-	-	72	33188.75	-	-	**74**	**25370.02**	-	-
9	72	33074.30	-	-	72	33350.51	-	-	72	24698.05	23	12301.63
10	-	-	-	-	72	31766.56	-	-	72	24324.76	23	11163.89
1000 customers												
1	100	53904.23	-	-	-	-	-	-	-	-	-	-
2	91	50701.78	-	-	-	-	-	-	-	-	-	-
3	91	46169.17	-	-	90	43390.58	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-	-	-
5	91	54032.44	-	-	90	46631.89	-	-	-	-	-	-
6	-	-	-	-	-	-	-	-	-	-	-	-
7	91	45729.79	-	-	-	-	-	-	**98**	**42824.09**	-	-
8	-	-	-	-	90	45406.46	-	-	**93**	**42499.59**	-	-
9	-	-	-	-	90	45149.72	-	-	**90**	**41318.12**	**29**	**16751.82**
10	-	-	-	-	90	44947.71	-	-	90	40586.60	-	-

Table 4.8 New best solution values

the TDR phase. Most important, we see that while some operators are more efficient than others, none is of second importance.

To conclude this section, we present another table to compare the effort spent by the algorithm in both phases. For each data set with 400 to 1000 customers and for each phase, Table 4.10 specifies the average number of LNS iterations (*nb itr*) performed (recall that 800 iterations in the TDR phase was set by a parameter), the computational time (in minutes) spent in this phase, and the average time per iteration (*time per it*). It also reports the average number of instances per data set (each containing 60 instances) for which the VNR phase was stopped because the lower bound m_{lb} on the number of vehicles was reached. From these results, we observe that the average number of iterations executed in the VNR phase is relatively low. This is due to the large number of times (more than 2/3 of the times) that this phase was stopped because the lower bound m_{lb} was reached. In fact, for most instances, the VNR phase is not very time consuming but, for others where it is difficult to reduce the number of vehicles used, it can take up to 50% of the total computational time. For the different sizes, the time spent in this phase varies between 11 and 22% of the overall time. This can be a bit surprising given the very good results that was obtained for the CNVs. Finally, we note also that, as expected, the average computational time per iteration increases with the size of the instances and that this average time is quite similar in both phases.

4.5.4 Sensitivity analysis

Tests were made to verify the behaviour of our algorithm with regards to variations of certain parameter values. For each instance size from 400 to 1000 customers, a subset *sub-(size)* of instances was selected to cover a variety of instance types. The subset *sub-400* consists of the following instances: R1_4_9, R2_4_1, RC1_4_4, RC2_4_2, RC2_4_5, RC2_4_6, C1_4_7, C1_4_8, C2_4_2 and C2_4_3. For the other sizes, the susbets contain the corresponding instances (R1_6_9, R2_6_1, ...).

The sensitivity analysis was performed for each parameter in Table 4.1, where one parameter value was changed at once. The test results can be found in Tables 4.11–4.16. As in Section 4.5.3, five runs were performed for each instance and, for each subset, two values are reported, namely, the mean number of vehicles used and the mean total distance. CNV and CTD values are given as well and correspond to the sums over all the subsets. For comparison purposes, the last column provides the values of the best published solutions. Finally, the last row of each table gives the average computation time (in minutes) for solving an instance.

The results reported in Tables 4.11–4.16 are discussed in the following paragraphs, where the emphasis is put on the CNV and CTD values, as well on the computational times. Indeed,

	Route portion				Proximity			
	VNR		TDR		VNR		TDR	
instance size	nb calls	nb impr	nb calls	nb impr	nb calls	nb impr	nb calls	nb impr
400	9479	2367	56962	13532	14191	4231	73094	17747
600	8298	2114	56347	13885	13814	4600	77922	19300
800	13651	3522	52284	13651	21481	8241	81312	22361
1000	13552	4563	56197	16683	23851	9277	79056	23851
Total	43056	12566	221790	57751	71166	26349	311384	83259
%		29.2		26.0		37.0		26.7
	Longest detour				Time			
	VNR		TDR		VNR		TDR	
instance size	nb calls	nb impr	nb calls	nb impr	nb calls	nb impr	nb calls	nb impr
400	12388	3761	52751	10697	11100	3501	57193	12438
600	9694	3341	47047	8251	11233	4307	58684	12388
800	15012	5616	45587	7764	15609	6492	60817	14031
1000	14949	6197	42044	8084	15937	7156	62703	16463
Total	52043	18915	187429	34796	53879	21456	239397	55320
%		36.3		18.6		39.8		23.1

Table 4.9 Use of different operators

instance	VNR				TDR		
size	nb it	tot time	time per it	m_{lb} reached	nb itr	tot time	time per it
400	158	9	0.06	40.2	800	81	0.10
600	144	13	0.09	44.8	800	92	0.11
800	214	26	0.12	38.8	800	103	0.13
1000	221	35	0.16	39.6	800	127	0.16

Table 4.10 Statistics on the two phases

in most cases, the results for the different subsets are very consistent with the cumulative values.

Number of customers removed (Table 4.11): The results indicate that using larger neighborhoods yields better solutions, but takes longer computational times. Removing 100 customers is however sufficient to obtain the same CNV as that of the best published solutions and requires much less computational time than the case where 120 cutomers are removed.

Maximum number of columns in memory pool (Table 4.12): To a certain extent, keeping more columns in the memory pool helps computing better solutions. It however requires longer computational times to manage a larger number of columns. Notice that keeping no columns at all in memory yields very bad quality solutions.

Number of tabu search iterations per column generation iteration in the VNR phase (Table 4.13): On average, allowing more tabu search iterations per column generation iteration in the VNR phase helps reducing the number of vehicles used. However, with $1.3|\mathcal{N}|$ tabu search iterations, it was not possible to find 1421 vehicles among the best solutions as it was the case with $1.0|\mathcal{N}|$ iterations. This is due to the heuristic nature of the method. It should be noted that increasing the value of this parameter slowly increases computational times, because the number of LNS iterations performed in the VNR phase accounts for approximately 11 to 17% of the total number of LNS iterations (see Table 4.10).

	80		100		120		
	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	20.8	20.90	20.7	20.88	20.7	20.86	20.6
	6551.27	6582.42	6587.39	6572.04	6574.22	6549.16	6651.14
sub-600	30.7	30.94	30.4	30.64	30.4	30.66	30.4
	13414.03	13332.73	13557.75	13462.11	13475.26	13413.95	13412.17
sub-800	40.6	40.94	40.5	40.86	40.5	40.74	40.6
	22536.34	22492.23	22379.96	22363.30	22305.19	22342.87	22398.83
sub-1000	50.6	51.04	50.5	50.80	50.5	50.72	50.5
	34446.49	34391.39	34261.47	34378.06	34147.35	34202.54	33694.66
CNV	1427	1438.2	1421	1431.8	1421	1429.8	1421
CTD	769481	767987	767866	767755	765020	765085	761568
Time (min)		88		143		218	

Table 4.11 Sensitivity analysis on the number of customers removed

	0		20000		50000		100000		
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	21.0	21.10	20.8	20.92	20.7	20.88	20.7	20.72	20.6
	6636.47	6691.53	6583.66	6592.47	6587.39	6572.04	6566.48	6598.17	6651.14
sub-600	31.0	31.06	30.8	30.94	30.4	30.64	30.5	30.72	30.4
	13334.99	13514.01	13318.26	13379.34	13557.75	13462.11	13407.66	13351.09	13412.17
sub-800	41.0	41.32	40.6	40.98	40.5	40.86	40.6	40.90	40.6
	22693.96	22691.46	22665.14	22568.92	22379.96	22363.30	22295.38	22299.63	22398.83
sub-1000	51.0	51.36	50.8	50.98	50.5	50.80	50.6	50.82	50.5
	34828.89	34902.62	34332.58	34615.52	34261.47	34378.06	34032.07	34237.34	33694.66
CNV	1440	1448.4	1430	1438.2	1421	1431.8	1424	1431.6	1421
CTD	774943	777996	768996	771563	767866	767755	763016	764862	761568
Time (min)		75		117		143		170	

Table 4.12 Sensitivity analysis on the maximum number of columns in memory pool

Number of tabu search iterations per initial solution in the TDR phase (Table 4.14): As expected, increasing the value of this parameter reduces the total distance. On the other hand, it increases computational times rather rapidly.

Maximum number of VNR iterations to find a feasible solution (Table 4.15): Increasing the value of this parameter can only improve the solution quality in terms of the number of vehicle used. With 800 iterations, we even succeeded to obtain 1420 vehicles which is better than the best results reported in the literature. Note also that increasing the value of this parameter slowly increases computational times, because the VNR phase is often stopped when the lower bound m_{lb} on the number of vehicles is reached.

Number of iterations in the TDR phase (Table 4.16): Increasing the value of this parameter cannot deteriorate the solution quality. In fact, it helps reducing the total distance as shown by the results. Computational times however increase rather rapidly with the number of iterations in the TDR phase.

To conclude this section, we would like to highlight the fact that our algorithm involves close to ten parameters, which might seem a lot to adjust carefully. However, most of them

	$0.7\|\mathcal{N}\|$		$1.0\|\mathcal{N}\|$		$1.3\|\mathcal{N}\|$		
	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	20.9	20.94	20.7	20.88	20.8	20.90	20.6
	6510.76	6539.91	6587.39	6572.05	6544.16	6550.11	6651.14
sub-600	30.6	30.84	30.4	30.64	30.4	30.70	30.4
	13329.61	13325.91	13557.75	13462.11	13494.42	13404.53	13412.17
sub-800	40.6	40.96	40.5	40.86	40.5	40.74	40.6
	22324.69	22289.15	22379.96	22363.30	22295.51	22386.59	22398.83
sub-1000	50.6	50.90	50.5	50.80	50.6	50.80	50.5
	34313.08	34347.92	34261.47	34378.06	34234.19	34341.86	33694.66
CNV	1427	1436.4	1421	1431.8	1423	1431.4	1421
CTD	764781	765029	767866	767755	765683	766831	761568
Time (min)		134		143		150	

Table 4.13 Sensitivity analysis on the number of tabu search iterations per column generation iteration in the VNR phase

	3		5		10		
	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	20.7	20.88	20.7	20.88	20.7	20.88	20.6
	6603.70	6594.60	6587.39	6572.05	6598.67	6582.22	6651.14
sub-600	30.4	30.64	30.4	30.64	30.4	30.64	30.4
	13579.14	13494.57	13557.75	13462.11	13548.48	13462.88	13412.17
sub-800	40.5	40.86	40.5	40.86	40.5	40.86	40.6
	22531.60	22513.82	22379.96	22363.30	22306.04	22333.11	22398.83
sub-1000	50.5	50.80	50.5	50.80	50.5	50.80	50.5
	34303.16	34450.05	34261.47	34378.06	34109.92	34282.22	33694.66
CNV	1421	1431.8	1421	1431.8	1421	1431.8	1421
CTD	770176	770530	767866	767755	765631	766604	761568
Time (min)		120		143		184	

Table 4.14 Sensitivity analysis on the number of tabu search iterations per initial solution in the TDR phase

are used to limit computational times in one way or another. The sensitivity analysis results presented in this section clearly show that our algorithm could have found better solutions if we have allowed more time to solve the instances. Hence, we had to make a compromise between solution quality and computational time. We believe that the parameter values used to produce the main results reported in Subsection 4.5.3 offer a good trade-off between these two criteria.

4.6 Conclusion

In this paper, we proposed a new LNS method for solving the VRPTW. This method takes advantage of the power of branch-and-price, the leading methodology for the exact solution of the VRPTW, to efficiently explore the neighborhoods. With this methodology, we succeded to find 145 new best solutions on the well-known 356 benchmark instances of Solomon (1987) and Gehring et Homberger (1999) and to obtain for all instance sizes the best

	400		600		800		
	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	20.7	20.90	20.7	20.88	20.6	20.82	20.6
	6596.57	6566.46	6587.39	6572.04	6632.94	6585.08	6651.14
sub-600	30.6	30.72	30.4	30.64	30.4	30.62	30.4
	13373.38	13410.48	13557.75	13462.11	13543.40	13458.55	13412.17
sub-800	40.5	40.90	40.5	40.86	40.5	40.80	40.6
	22350.63	22318.69	22379.96	22363.30	22384.03	22439.04	22398.83
sub-1000	50.6	50.90	50.5	50.80	50.5	50.78	50.5
	34218.14	34298.80	34261.47	34378.06	34295.44	34427.53	33694.66
CNV	1424	1434.4	1421	1431.8	1420	1430.2	1421
CTD	765387	765944	767866	767755	768558	769102	761568
Time (min)		131		143		152	

Table 4.15 Sensitivity analysis on the maximum number of VNR iterations to find a feasible solution

	400		800		1200		
	Best	Avg	Best	Avg	Best	Avg	Best pub
sub-400	20.7	20.88	20.7	20.88	20.7	20.88	20.6
	6629.36	6604.31	6587.39	6572.05	6580.95	6559.11	6651.14
sub-600	30.4	30.64	30.4	30.64	30.4	30.64	30.4
	13683.35	13577.36	13557.75	13462.11	13486.45	13402.23	13412.17
sub-800	40.5	40.86	40.5	40.86	40.5	40.86	40.6
	22618.20	22687.86	22379.96	22363.30	22289.87	22225.88	22398.83
sub-1000	50.5	50.80	50.5	50.80	50.5	50.80	50.5
	34575.54	34804.44	34261.47	34378.06	34114.23	34189.77	33694.66
CNV	1421	1431.8	1421	1431.8	1421	1431.8	1421
CTD	775064	776740	767866	767755	764715	763770	761568
Time (min)		100		143		185	

Table 4.16 Sensitivity analysis on the number of iterations in the TDR phase

cumulative number of vehicles (CNVs) and cumulative total distances (CTDs) compared to the results of the best know methods. Furthermore, we demonstrated through a sensitivity analysis on the parameters limiting computational time that our method could produce better solutions if additional computational time was used. Hence, the parameter setting used for our experiments offered a compromise between solution quality and computational time. This compromise led to acceptable computational times which are not as fast as those of certain other leading methods.

Combining branch-and-price and LNS is a relatively new idea that can be applied for a wide variety of vehicle and crew scheduling problems. This paper has shown that such a hybrid methodology can perform very well against the leading heursitic methods for a well-studied problem.

4.7 Nouvelles méthodes

Depuis la publication de cet article, un certain nombre de méthodes ont été appliquées au VRPTW. Plusieurs ont obtenu de très bons résultats, il est donc important de les mentionner. Le tableau 4.17 présente un résumé de ces résultats pour l'ensemble des instances de Solomon (1987) et Gehring et Homberger (1999) par taille de problèmes. NBD représente l'algorithme mémétique développé par Nagata *et al.* (2010), LZ, la méthode d'ascension de collines (hill climbing) avec chaînes d'éjections de Lim et Zhang (2007) et RTI, l'algorithme évolutif de Repoussis *et al.* (2009). PDR, pour Prescott-Gagnon *et al.*, représente notre méthode. Les temps de calcul sont en minutes et s'il existe un multiplicateur, par exemple, 5.0 x 5, c'est donc qu'il y a eu 5 tests d'une moyenne de 5 minutes et qu'une moyenne des meilleurs résultats trouvés sur les 5 tests pour chaque instance est présentée. À noter que Nagata *et al.* (2010) présentent deux variantes de sa méthode, une avec 5 tests par instance et une autre avec 1 seul test par instance mais où ils laissent plus de temps à leur méthode.

		PDR		LZ	RTI	NBD
Taille		Best	Avg	(2009)	(2009)	(2010)
100	CNV	405	406.6	405	405	**405**
	CTD	57240	57101	57368	57216	**57187**
	Temps (min)		30 x 5	38.5	53.7	5.0 x 5
200	CNV	694	695.0	694	694	**694**
	CTD	168556	168786	169296	169163	**168067**
	Temps (min)		53 x 5	93.2	96.3	4.1 x 5
400	CNV	1385	1388.0	1382	1381	**1381**
	CTD	389011	390771	393695	395936	**388466**
	Temps (min)		89 x 5	296	193	16.2 x 5
600	CNV	2071	2074.4	2068	**2066**	2067
	CTD	800797	805325	802681	**816326**	789420
	Temps (min)		105 x 5	647	289	80.4
800	CNV	2745	2750.6	2742	2739	**2738**
	CTD	1391344	1401549	1372427	1424321	**1357695**
	Temps (min)		129 x 5	1269	385	27.6 x 5
1000	CNV	3432	3437.8	3429	3428	**3424**
	CTD	2096823	2110187	2071643	2144830	**2040661**
	Temps (min)		162 x 5	1865	482	186
CPU			OPT 2.3GHz	P4 2.8GHz	P4 3GHz	OPT 2.4GHz

Tableau 4.17 Nouvelles méthodes pour le VRPTW

Hashimoto et Yagiura (2008) introduisent aussi une nouvelle méthode évolutive mais ils se servent de la meilleure valeur du nombre de véhicules pour chaque instance comme

information de base pour leur algorithme. En pratique, il n'est pas possible de connaître cette valeur ce qui donne ainsi un avantage à leur méthode. Pour cette raison, leurs résultats ne sont pas présentés dans le tableau 4.17.

CHAPITRE 5

EUROPEAN DRIVER RULES IN VEHICLE ROUTING WITH TIME WINDOWS

Article publié dans *Transportation Science* 44(4), 455–473, 2010 et écrit par:

ERIC PRESCOTT-GAGNON

École Polytechnique de Montréal

GUY DESAULNIERS

École Polytechnique de Montréal

MICHAEL DREXL

Fraunhofer - Center for Applied Research on Supply Chain Services SCS

LOUIS-MARTIN ROUSSEAU

École Polytechnique de Montréal

Abstract

As of April 2007, the European Union has new regulations concerning driver working hours. These rules force the placement of breaks and rests into the vehicle routes when consecutive driving or working time exceeds certain limits. This paper proposes a large neighborhood search method for the vehicle routing problem with time windows and driver regulations. In this method, the neighborhoods are explored using a column generation heuristic that relies on a tabu search algorithm for generating new columns (routes). Checking route feasibility after inserting a customer into a route in the tabu search algorithm is not an easy task. To do so, we model all feasibility rules as resource constraints, develop a label-setting algorithm to perform this check, and show how it can be used efficiently to validate multiple customer insertions into a given existing route. We test the overall solution method on modified Solomon's (1987) instances and report computational results that clearly show the efficiency of our method compared to two other existing heuristics.

Keywords: Vehicle routing, time windows, driver rules, large neighborhood search, heuristic column generation, resource constraints.

5.1 Introduction

Vehicle routing consists of determining a set of vehicle routes to service a set of customers at minimum cost. Applications of vehicle routing are numerous, so are the problem variants (see Toth et Vigo, 2002; Golden *et al.*, 2008). One of the first variants to appear in the literature is the vehicle routing problem with time windows (VRPTW) that can be briefly defined as follows. Given a set of customers, each with a known demand and a time window in which service must begin, and an unlimited set of identical vehicles with a fixed capacity that are all housed in a single depot, the VRPTW consists of finding vehicle routes such that each customer is serviced within its time window, each route starts and ends at the depot and respects vehicle capacity, the number of vehicles used is minimized as a primary objective, and the total distance traveled is minimized as a secondary objective. The VRPTW has been well studied in the literature (see Kallehauge *et al.*, 2005; Bräysy et Gendreau, 2005a,b).

As of April 2007, new regulations for driving hours have been enforced in the European Union. These rules are defined in Regulation (EC) No 561/2006 of the European Union (2006). Furthermore, European drivers are also subject to additional regulations concerning their working hours (which include driving and servicing hours) as stipulated in Directive 2002/15/EC of the European Union (2002). All these rules strongly restrict the feasibility of vehicle routes that last several days and are assigned to a single driver. In this case, two approaches can be used to construct vehicle routes. In a two-phase approach, routes are first built without considering driver rules and are later modified to take these rules into account. In an integrated approach, the driver rules are directly addressed while constructing

the vehicle routes, which is usually more complex but provides lower-cost solutions. In this paper, we propose a hybrid heuristic combining local search and mathematical programming for solving the integrated problem of building vehicle routes that respect time windows and all driver rules dictated by Regulation (EC) No 561/2006 and Directive 2002/15/EC applicable to routes lasting less than one week. We call this problem the VRPTW with driver rules (VRPTWDR).

The literature on vehicle routing that includes driver regulations is rather scant. Savelsbergh et Sol (1998) proposed a branch-and-price algorithm for a pickup and delivery problem where breaks for drivers must be taken within a time interval around noon. Xu *et al.* (2003) tackled a pickup and delivery problem with several complicating side constraints including driving time restrictions that force night rests. To solve this problem, they developed a column generation method that relies on a heuristic column generator. Bartodziej *et al.* (2009) designed a column generation method and three metaheuristics for solving a combined vehicle and crew scheduling problem with time windows and rest regulations. Goel et Gruhn (2006) and Goel (2009) proposed a large neighborhood search framework for solving a variant of the VRPTWDR that considers only a subset of the European regulations. The chosen rules are strict enough to ensure the feasibility of the routes when all regulations are taken into account. Zäpfel et Bögl (2008) developed a two-phase algorithm for a complex VRPTW with certain driver rules. In the first phase, a vehicle routing problem is solved using a metaheuristic, while in the second phase, a personnel assignment problem is solved using a simple heuristic. Daily constraints (breaks and maximum daily driving time) are tackled by the routing algorithm. Weekly constraints (rests and maximum weekly driving time) are treated in the personnel assignment problem. In a very recent working paper, Kok *et al.* (2010) presented a heuristic dynamic programming algorithm that can solve the VRPTWDR (with all European regulations). They obtained much better solutions than Goel (2009) for the same instances (considering only a subset of the driver rules). They also showed that considering complex regulations that provide additional flexibility for positioning breaks and rests can yield substantial cost savings.

This paper introduces a large neighborhood search algorithm for the VRPTWDR. This algorithm uses a column generation heuristic for exploring a neighborhood (that is, reoptimizing the current solution). In this heuristic, columns are generated by a tabu search that checks the feasibility of the driver rules using resource constraints. Computational results obtained on the instances used by Goel (2009) and Kok *et al.* (2010) show that our algorithm clearly outperforms their algorithms, which are the only ones proposed so far in the literature for the VRPTWDR.

This paper is structured as follows. Section 5.2 defines the VRPTWDR. Section 5.3

describes the proposed large neighborhood search algorithm, while Section 5.4 details the procedure used to check the feasibility of a route. Section 5.5 reports the results of our computational experiments. Finally, conclusions are drawn in Section 5.6.

5.2 Problem statement

Given a set of identical vehicles assigned to a single depot, the VRPTW consists of computing a set of feasible routes (one per vehicle used) to deliver goods to a set $\mathcal{N}$ of customers while minimizing first the total number of vehicles used and second the total distance traveled. Each customer $i \in \mathcal{N}$ must be visited by exactly one vehicle to receive a quantity q_i of goods and its service must begin within a prescribed time window $[a_i, b_i]$. A route starts and ends at the depot and is feasible if it respects the time windows of the visited customers and the total quantity delivered does not exceed the vehicle capacity Q.

Denote the depot by 0 and let $\mathcal{N}_0 = \mathcal{N} \cup \{0\}$. Without loss of generality, we associate an unrestrictive time window $[a_0, b_0] = [0, H]$ with the depot, where H is the length of the planning horizon. Let t_{ij}, $i, j \in \mathcal{N}_0$, be the driving time between locations i and j, and s_i be the service time at i, $i \in \mathcal{N}_0$ ($s_0 = 0$). Furthermore, let c_{ij} be the travel distance between i and j (which is usually proportional to t_{ij}). We assume that the driving times and the travel distances satisfy the triangle inequality. In this case, customer j can be visited immediately after customer i only if $a_i + s_i + t_{ij} \leq b_j$. Note that, although the service at customer j must start within time window $[a_j, b_j]$, the vehicle servicing this customer can arrive earlier than a_j and wait until a_j before starting service.

The VRPTWDR extends the VRPTW with the additional constraint that every selected route must satisfy the driver regulations defined below. These regulations impose scheduling breaks and rests for the drivers along the vehicle routes. However, these breaks and rests cannot interrupt the service at a customer (customer service is not preemptive).

This problem definition relies on the assumption that every vehicle is assigned to a single driver for the whole 6-day horizon as it is often the case in national and international road transportation (long distances between the customers). This allows to impose directly the driver regulations on the vehicle routes. Furthermore, as in Goel (2009) and Kok *et al.* (2010), we assume that the VRPTWDR is defined over a 6-day horizon (from Monday 00:00 to Saturday 24:00), that is, $H = 8640$ minutes. Indeed, in Europe, drivers are not allowed to work on Sundays and their weekly rests (to be defined below) usually contain Sunday.

5.2.1 Definitions for driver regulations

The following definitions are needed to describe the driver regulations. Each period of time in the following statements shall be uninterrupted.

- A calendar *week* is defined by the period of time between 00:00 on Monday through 24:00 on Sunday.
- A *break* is any period of time of at least 15 minutes but less than 3 hours where a driver can dispose freely of his time.
  - A *short break* is a break of at least 15 minutes but less than 45 minutes.
  - A *long break* is a break of at least 45 minutes, or a break of at least 30 minutes if a short break was taken since the last long break or rest.
- A *rest* is any period of time of at least 3 hours where a driver can dispose freely of his time.
  - A *short rest* is a rest of at least 3 hours but less than 9 hours.
  - A *long rest* is a rest of at least 9 hours but less than 24 hours.
  - A *regular daily rest* is either a long rest of at least 11 hours or a combination of a short rest and a long rest.
  - A *reduced daily rest* is a long rest of less than 11 hours not preceded by a short rest.
  - A *weekly rest* is a rest of at least 24 hours. In our case, we assume that a weekly rest is taken before the start and after the end of every route.
- The *interval driving time* is the total cumulated driving time between two long breaks or rests.
- The *daily driving time* is the total cumulated driving time between two regular or reduced rests (daily or weekly).
- The *weekly driving time* is the total cumulated driving time in a calendar week.
- The *interval working time* is the total cumulated working time between two long breaks or rests. In our case, working time is cumulated when driving or servicing a customer.
- The *weekly working time* is the total cumulated working time in a calendar week.

5.2.2 Regulations on driving time and working time

There are five different driving or working time periods, each having its own restrictions.

- The interval driving time must not exceed 4.5 hours.
- The daily driving time must not exceed 9 hours. It is possible to increase the daily driving time to a maximum of 10 hours but not more than twice per calendar week.
- The weekly driving time must not exceed 56 hours.
- The interval working time must not exceed 6 hours.

– The weekly working time must not exceed 60 hours.

These last two regulations are the only ones imposed by Directive 2002/15/EC of the European Union (2002). All the other rules, including those stated in the next subsection, are enforced by Regulation (EC) No 561/2006 of theEuropean Union (2006).

Note that, for the interval working time rule, Directive 2002/15/EC is less constraining than what we stated above. In fact, a working interval can be interrupted by a 30-minute break (not necessarily a long break) if the daily working time does not exceed 9 hours, otherwise a 45-minute (long) break or a rest is needed. In any case, such a break may be subdivided into periods of at least 15 minutes each. In a context where the distances between the customers are rather long (such as the one we consider in our experiments), the interval working time rule is often redundant with the interval driving time rule. This justifies our choice to consider a more restrictive rule on the interval working time.

5.2.3 Regulations on rest periods

The following rules apply to the daily rests.

– Within 24 hours after the end of the previous daily rest or weekly rest, a driver must have taken a new daily rest.
– If the portion of the daily rest period which falls within that 24-hour period is at least 9 hours but less than 11 hours, then the daily rest in question shall be regarded as a reduced daily rest.
– The daily rest must not necessarily be over at the end of the 24-hour period as long as more than 11 (or 9) hours of daily rest fall within the 24-hour period. In other words, a daily rest must start no later than 13 hours after the end of the last one in case of a regular daily rest or 15 hours in case of a reduced daily rest. These values (13 and 15 hours) are referred to as the maximum daily duration and the extended maximum daily duration.
– A maximum of 3 reduced daily rests can be taken between two weekly rests.

The following rule applies to the weekly rests.

– There must be no more than six 24-hour periods (144 hours) between two weekly rests. This rule is, thus, directly taken into account by considering a 6-day horizon and weekly rests before and after each route.

5.3 Solution method

For solving the VRPTW, Prescott-Gagnon *et al.* (2009) introduced a large neighborhood search (LNS) heuristic that takes advantage of the power of branch-and-price (BP), a leading

methodology for the exact solution of the VRPTW. The LNS heuristic chooses, at each iteration, between different neighborhood structures to destroy parts of a current solution and uses a BP heuristic to resconstruct these parts, potentially yielding an improved solution. Compared to the best known methods, this algorithm produced better solutions, especially on large instances where the number of vehicles used was significantly reduced.

For the VRPTWDR, we propose to use a similar LNS heuristic. In fact, the framework is identical: the same neighborhood-defining operators are applied to destroy the current solution, a BP heuristic that relies on a tabu search (TS) column generator is invoked to construct a new solution, and this TS algorithm allows two possible move types to modify a route under construction, namely, the deletion of a customer from this route and the insertion of a new customer into it. Deleting a customer from a route is easy, either in the VRPTW or the VRPTWDR. However, inserting a customer into a route requires checking the feasibility of the resulting route. This feasibility check can be done fairly easily for the VRPTW. This task is much more complex for the VRPTWDR. Hence, the algorithmic contribution of this paper is to develop an efficient heuristic procedure to check the feasibility of a route after a customer insertion. This procedure uses labels with resource components to represent partial routes and their corresponding breaks and rests as well as resource extension functions (REFs) to propagate these labels.

Figure 5.1 illustrates the link between the various components of the proposed solution approach. Starting from an initial solution, an LNS heuristic alternating between a destruction phase and a reconstruction phase is applied to improve it. Reconstruction is performed using a BP heuristic that solves at each iteration a linear relaxation by column generation before fixing a route if the linear relaxation solution is fractional. In a column generation algorithm, a restricted master problem is solved at each iteration before solving a subproblem. In our method, we solve the subproblem using a TS heuristic. In this heuristic, customers are removed or inserted into a current route. The proposed insertion feasibility checker based on REFs is invoked in this TS heuristic.

In this section, we briefly describe the components of this solution method that are common with the method of Prescott-Gagnon *et al.* (2009). The reader is referred to Prescott-Gagnon *et al.* (2009) for further details. The insertion feasibility checker will be detailed in Section 5.4.

5.3.1 Large neighborhood search

Given an initial solution that consists of a set of routes covering all customers, an LNS heuristic is an iterative method that destroys parts of a current solution at each iteration using a neighborhood-defining operator and constructs a new solution using an optimization

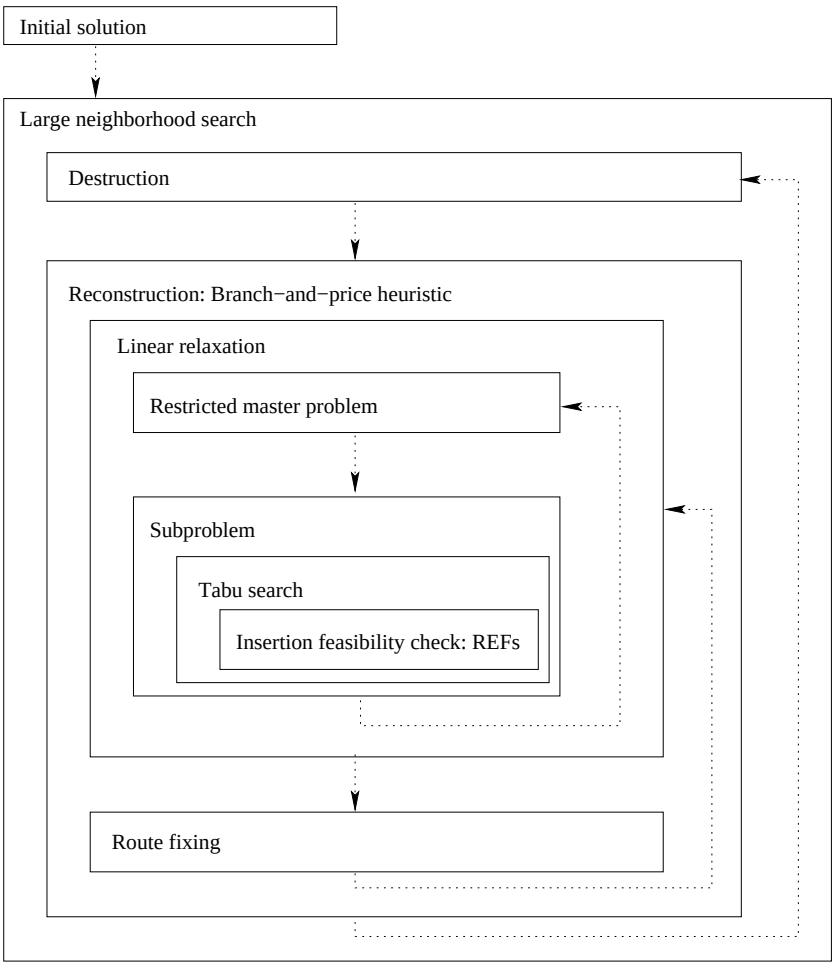

Figure 5.1 Algorithm framework

algorithm. In our case, a neighborhood-defining operator selects a predefined number of customers to remove (denoted M^{rem}) from the current solution, leaving a partial solution composed of partial routes that remain fixed in the construction step. Four such operators were used: a proximity operator that selects, one at a time, customers that are related geographically and temporally; a route portion operator similar to the proximity operator but that also removes for each selected customer some of its adjacent neighbors in its current route; a longest detour operator that removes the customers yielding the largest distance increases for servicing them; and a time operator that simply selects customers that are serviced almost at the same time. The operator applied at each iteration to destroy the current solution is chosen using a roulette-wheel procedure that favors the operators that have yielded improved solutions in the past iterations.

In the construction step, the BP heuristic described in the next subsection is executed to build a solution that contains the partial routes kept from the previous solution. The

computed solution always becomes the current solution even when its cost is greater than the cost of the previous solution.

The LNS algorithm proceeds in two phases. The first phase aims at minimizing the total number of vehicles used. To do so, it imposes an upper bound m_{ub} on this number of vehicles while allowing to not visit the customers at the expense of a large penalty for each unserviced customer. When the algorithm succeeds to find a solution that services all customers within a limited number of iterations (denoted I_1^{max}), m_{ub} is lowered by one and the search for a new feasible solution is started again. When it fails to do so, m_{ub} is increased by one (that is, to the number of vehicles used in the last feasible solution found) and the second phase is triggered. In this second phase, the algorithm performs a fixed number of iterations (denoted I_2^{max}) to reduce the total distance traveled.

5.3.2 Branch-and-price heuristic

At each LNS iteration, a BP heuristic is used to build a new solution that preserves the fixed parts of the current solution. Such a heuristic consists of a heuristic column generation method (to compute linear relaxation solutions) embedded into a heuristic branch-and-bound method (to derive integer solutions). It relies on the following formulation of the VRPTWDR.

Let Ω be the subset of all the feasible routes that respect the fixed parts of the current solution. With each route $p \in \Omega$, associate the following parameters: c_p is its cost and $v_{ip}, \forall i \in \mathcal{N}$, takes value 1 if customer i is serviced by route p and 0 otherwise. Finally, define a binary variable θ_p for each route $p \in \Omega$ indicating whether or not route p is part of the new solution. With this notation, the VRPTWDR (restricted by the fixed route parts) can be formulated as follows:

$$\text{Minimize} \qquad \sum_{p \in \Omega} c_p \theta_p \tag{5.1}$$

$$\text{subject to:} \qquad \sum_{p \in \Omega} v_{ip} \theta_p = 1, \qquad \forall\, i \in \mathcal{N} \tag{5.2}$$

$$\sum_{p \in \Omega} \theta_p \leq m_{ub}, \tag{5.3}$$

$$\theta_p \text{ binary}, \qquad \forall\, p \in \Omega. \tag{5.4}$$

The objective function (5.1) aims at minimizing total cost. Set partitioning constraints (5.2) ensure that each customer is visited exactly once by one vehicle. Note that, in the first phase of the LNS algorithm, slack variables (highly penalized in the objective function)

are introduced in these constraints to allow customer uncovering. Constraint (5.3) imposes an upper bound m_{ub} on the number of vehicles that can be used. Recall that this upper bound varies during the first phase of the LNS algorithm and is fixed in the second phase (see Section 5.3.1). Finally, binary requirements on the variables are expressed by (5.4).

In a column generation context, the linear relaxation of model (5.1)–(5.4) is called the master problem. Column generation is an iterative process that solves at each iteration the master problem restricted to a subset of the variables, then called the restricted master problem (RMP). The dual solution of this RMP is then transfered to a subproblem whose objective is to generate negative reduced cost variables to be added into the current RMP. The latter is then solved again with the augmented subset of variables. When no negative reduced cost variables exist, column generation stops and the computed solution of the current RMP is also optimal for the master problem.

The column generation subproblem consists of finding a least reduced cost route. Let α_i, $i \in \mathcal{N}$, and μ be the dual variable values of the RMP constraints (5.2) and (5.3) at the current column generation iteration, respectively. The reduced cost $\bar{c}_p$ of a feasible route $p = i_0 - i_1 - i_2 - \ldots - i_k - i_{k+1}$ starting and ending at the depot $i_0 = i_{k+1} = 0$ and visiting k customers $i_1, i_2, \ldots, i_k$ is computed as

$$\bar{c}_p = \sum_{\ell=0}^{k} \bar{c}_{i_\ell i_{\ell+1}},$$

where

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \alpha_i & \text{if } i \in \mathcal{N} \\ c_{ij} - \mu & \text{if } i = 0. \end{cases}$$

This subproblem can be modeled as an NP-hard resource-constrained elementary shortest path problem (see Irnich et Desaulniers, 2005). In the proposed column generation heuristic, we solve it using the TS heuristic described in the next subsection. Because this TS heuristic might not find a negative reduced cost route when such a route exists, column generation may terminate prematurely, possibly yielding a suboptimal solution for the master problem.

In order to quickly derive an integer solution, we use a heuristic branch-and-bound method that explores a single branch (that is, no backtracking is performed). After solving a linear relaxation for which a fractional solution was computed, the route variable θ_p with the largest fractional value is simply fixed at 1, defining a new linear relaxation to solve (or, equivalently, a new branch-and-bound node to explore).

5.3.3 Tabu search column generator

To solve the column generation subproblem, we use a TS heuristic that can often generate negative reduced cost columns (routes) in fast computational times. Such a heuristic is an iterative method that starts from an initial solution and applies moves to potentially improve it. In our case, a solution is a route and two types of moves are considered, namely, inserting a customer into the route and removing a customer from the route. At each iteration, the move creating the best solution is chosen even if the objective value deteriorates. To avoid cycling, a tabu list is used in order to forbid recent moves to be reversed for a given number of iterations, allowing the search to escape from local minima.

In the TS algorithm, the sequences of customers (partial routes) fixed in the destruction phase of the LNS algorithm are treated as aggregated customers meaning that new customers cannot be inserted within a sequence. Also, the search space is limited to feasible routes. Thus, for each insertion move, route feasibility is checked using the procedure described in Section 5.4. No feasibility check needs to be performed for customer removal moves (under the assumption that the travel times respect the triangle inequality).

To diversify the search, the TS heuristic is started multiple times from a set of different initial solutions and limited to a maximum number of iterations for each initial solution. In the first phase of the LNS method, the routes associated with the positive-valued variables in the current RMP solution form the set of initial solutions. In the second phase, this set contains the routes associated with all basic variables. In both cases, all initial solutions have a reduced cost of 0, and are thus very good initial solutions since we are looking for negative reduced cost solutions.

5.4 Route feasibility check

In the TS column generator presented above, one needs to validate that each customer insertion yields a route that can be operated by a vehicle and its driver. This validation requires, among others, the placement of breaks and rest periods along the route in accordance with the regulations described in Section 5.2.

In this section, we present a labeling algorithm that checks route feasibility and show how it can be used in conjunction with a filtering algorithm to assess efficiently the feasibility of multiple insertions into a given feasible route. This algorithm relies on resource constraints to model all route feasibility rules(see Irnich et Desaulniers, 2005). It uses a total of 15 resources. However, to simplify its description, we consider in this section only a subset of the feasibility rules and options that requires 8 resources. These rules are the time windows, the vehicle capacity, the maximum interval driving time, the maximum daily driving time,

and the maximum daily duration. Furthermore, among all options (short breaks, short rests, reduced rests, and extended maximum daily driving time), we consider only the possibility of using short breaks. How to deal with the other rules and options is exposed in Appendix A.

5.4.1 Labeling algorithm

Let $p = i_0 - i_1 - i_2 - \ldots - i_k - i_{k+1}$ be a route starting and ending at the depot $i_0 = i_{k+1} = 0$ and visiting the k customers $i_1, i_2, \ldots, i_k$ (including the inserted customer). Associate a path with this route where i_ℓ, $\ell \in \{0, 1, \ldots, k+1\}$, are the vertices of this path (two different vertices for the depot) and $(i_\ell, i_{\ell+1})$, $\ell \in \{0, 1, \ldots, k\}$, its arcs. Denote by $\mathcal{V}_p$ and $\mathcal{A}_p$ these vertex and arc sets, respectively. To check the feasibility of such a path (route), we use a labeling algorithm that starts with an initial label at vertex 0 and propagates this label forwardly along the arcs of this path to create new labels. A label represents a feasible partial path (which implicitly includes breaks and rests on the arcs) originating from vertex 0 and is given by a resource vector. Such a vector E_i is associated with the destination vertex $i \in \mathcal{V}_p$ of the partial path and contains 8 components, one for each resource (all times are expressed in minutes).

$T_i \in [a_i, b_i]$: earliest service start time at vertex i.

$L_i \in [0, Q]$: accumulated demand (load) up to vertex i (including it).

$T_i^{\text{drive,int}} \in [0, 270]$: interval driving time up to vertex i.

$n_i^{\text{sb}} \in [0, 1]$: number of short breaks taken before vertex i since the last long break or rest.

$T_i^{\text{drive,day}} \in [0, 540]$: daily driving time up to vertex i.

$LT_i \in [a_i, b_i]$: latest start of service time at vertex i that does not yield unnecessary waiting or, equivalently, earliest start of service time if the current day starts as late as possible. It is used to compute the daily duration.

$XT_i \in [0, \infty]$: extended latest start of service time at vertex i if there were no time windows at this vertex. It is also used to compute the daily duration.

$D_i \in [0, 780]$: current daily duration up to vertex i.

Thus, $E_i = (T_i, L_i, T_i^{\text{drive,int}}, n_i^{\text{sb}}, T_i^{\text{drive,day}}, LT_i, XT_i, D_i)$. Let $\mathcal{R}$ be the set of resources. Each resource $r \in \mathcal{R}$ is constrained by a resource window $[a_i^r, b_i^r]$ at each vertex $i \in \mathcal{V}_p$. These windows were presented above. A label represents a feasible partial path if the value of each of its resource components falls into its corresponding resource window. If this is the case,

we say that the label (or the resource vector) is feasible.

The pseudo-code of the label-setting algorithm is given in Algorithm 5.1, where $\mathcal{E}(i)$ denotes the current subset of untreated labels at vertex $i \in \mathcal{N}_p$. The algorithm starts by defining an initial label E_{i_0} associated with vertex i_0 and initializing the sets $\mathcal{E}(i_\ell)$ for all $\ell \in \{0, 1, \ldots, k+1\}$. Then, in the main loop (Steps 3–7), it selects at each iteration an untreated label and extends it to generate new labels. The label extension is performed using the procedure described in Subsection 5.4.2 and may consider various possibilities (no break nor rest, short break, long break, daily rest, and combinations of these) in addition to traveling from i_0 to i_1. This loop stops when one feasible label is created at vertex i_{k+1} or when all existing labels have been treated. In the hope of meeting the former stopping criterion as soon as possible, the labels are treated in a depth-first order (Steps 4–5).

Algorithm 5.1 Exact route feasibility check algorithm

Require: Route (path) $p = i_0 - i_1 - i_2 - \ldots - i_k - i_{k+1}$

1: $E_{i_0} \leftarrow (0, 0, 0, 0, 0, H, H, 0)$
2: $\mathcal{E}(i_0) \leftarrow \{E_{i_0}\}$, $\mathcal{E}(i_\ell) \leftarrow \emptyset, \forall\, \ell \in \{1, 2, \ldots, k+1\}$
3: **while** $\mathcal{E}(i_{k+1}) = \emptyset$ and $\bigcup_{\ell=0}^{k} \mathcal{E}(i_\ell) \neq \emptyset$ **do**
4: $\quad \ell^* \leftarrow \max_{\ell \in \{0,1,\ldots,k\}} \{\ell \mid \mathcal{E}(i_\ell) \neq \emptyset\}$
5: $\quad$ Select a label $E_{i_{\ell^*}}$ in $\mathcal{E}(i_{\ell^*})$
6: $\quad \mathcal{E}(i_{\ell^*+1}) \leftarrow \mathcal{E}(i_{\ell^*+1}) \cup ExtendLabel(E_{i_{\ell^*}}, i_{\ell^*}, i_{\ell^*+1})$
7: $\quad \mathcal{E}(i_{\ell^*}) \leftarrow \mathcal{E}(i_{\ell^*}) \setminus \{E_{i_{\ell^*}}\}$
8: **if** $\mathcal{E}(i_{k+1}) \neq \emptyset$ **then**
9: $\quad$ Return p is feasible
10: **else**
11: $\quad$ Return p is infeasible

5.4.2 Label extension

At each iteration of Algorithm 5.1, a label E_i is extended along an arc $(i, j) \in \mathcal{A}_p$, where $i = i_{\ell^*}$ and $j = i_{\ell^*+1}$. This label represents a partial path ending at vertex i that we want to extend along the arc (i, j). This extension includes the time of service at vertex i and the travel time between vertices i and j. It may also include breaks and rests. In fact, several combinations of breaks and rests may yield a feasible label at vertex j.

To generate all feasible labels from E_i, we adopt the following strategy. First, we extend E_i assuming that no break, nor rest will be taken along arc (i, j). Then, we go over all possible options involving the placement of breaks and rests along this arc. These options are: insert a short break, insert a long break, and insert a regular long rest.

Using resource extension functions, the first extension yields a new resource vector E_j that might not be feasible at vertex j with respect to certain driver rules. This resource vector is then modified by other resource extension functions that are specific to each selected option. All these resource extension functions are described next.

Resource extension functions

Given a feasible label $E_i = (T_i, L_i, T_i^{\text{drive,int}}, n_i^{\text{sb}}, T_i^{\text{drive,day}}, LT_i, XT_i, D_i)$ at vertex i, it is first extended along the arc $(i,j) \in \mathcal{A}_p$ assuming that only the service at vertex i and the traveling between i and j are realized along this arc. This extension is performed using the following resource extension functions to generate a new resource vector $E_j = (T_j, L_j, T_j^{\text{drive,int}}, n_j^{\text{sb}}, T_j^{\text{drive,day}}, LT_j, XT_j, D_j)$.

$$T_j = T_i + s_i + t_{ij} \tag{5.5}$$

$$L_j = L_i + q_j \tag{5.6}$$

$$T_j^{\text{drive,int}} = T_i^{\text{drive,int}} + t_{ij} \tag{5.7}$$

$$n_j^{\text{sb}} = n_i^{\text{sb}} \tag{5.8}$$

$$T_j^{\text{drive,day}} = T_i^{\text{drive,day}} + t_{ij} \tag{5.9}$$

$$XT_j = LT_i + s_i + t_{ij} \tag{5.10}$$

$$LT_j = \max\{\min\{XT_j, b_j\}, a_j\} \tag{5.11}$$

$$D_j = D_i + LT_j - LT_i + \max\{XT_j - b_j, 0\} \tag{5.12}$$

These resource extension functions are denoted $REF_{first}(\cdot,\cdot,\cdot)$, that is, $E_j = REF_{first}(E_i, i, j)$. The component T_j, as computed by (5.5), indicates the earliest arrival time at vertex j, which can be less than a_j. When creating a label in which this component must represent the earliest start of service time at vertex j, we use the function $AdjTime(E_j, j)$ that duplicates all components except the T_j component which is replaced by $\max\{T_j, a_j\}$.

The computation of the current daily duration D_j relies on the components XT_j and LT_j. It assumes that the current day starts as late as possible (that is, the preceding rest has been extended to a maximum). Under this assumption, LT_j provides the earliest start of service time at vertex j. To compute LT_j using (5.11), one needs to compute first what would be this time if there were no time window at vertex j (given by XT_j). If $XT_j < a_j$, then waiting $a_j - XT_j$ minutes before starting service at vertex j is unavoidable and LT_j is set to a_j. If $XT_j \in [a_j, b_j]$, then $LT_j = XT_j$. If $XT_j > b_j$, then the current day must begin earlier (by shortening the preceding daily rest by $XT_j - b_j$ minutes if possible) and LT_j is set to b_j. Figure 5.2 illustrates these three cases. In the first case, the start time of the day does not

need to be changed and the daily duration D_j can be computed as $D_i + s_i + t_{ij} + a_j - XT_j = D_i + LT_j - LT_i$. In the second case, the start time of the day also remains the same, there is no additional waiting, and D_j computes as $D_i + s_i + t_{ij} = D_i + LT_j - LT_i$. Finally, in the third case, the day start time is pushed backward by $XT_j - b_j$ minutes and D_j is given by $D_i + s_i + t_{ij} + XT_j - b_j = D_i + LT_j - LT_i + \max\{XT_j - b_j, 0\}$. Note that shifting backward by $XT_j - b_j$ the start time of a day is always feasible unless $T_j > b_j$, in which case the label is declared infeasible and discarded.

A computed resource vector E_j might not be feasible with regards to the driver regulations. However, if $T_j < b_j$ and $L_j \leq Q$, adding breaks and rests along the arc (i, j) might yield a feasible label. Furthermore, even if all driver regulations are met, adding a break or a rest might be advantageous when unavoidable waiting occurs. In these cases, three options can be considered: inserting a short break (sb), a long break (lb), or a regular long rest (lr). Given a resource vector $E_j = (T_j, L_j, T_j^{\text{drive,int}}, n_j^{\text{sb}}, T_j^{\text{drive,day}}, LT_j, XT_j, D_j)$, the subset of options applicable along an arc $(i, j) \in \mathcal{A}_p$ is denoted $\Delta(E_j, j) \subseteq \{sb, lb, lr\}$. Applying such an option generates a new resource vector $\overline{E_j} = (\overline{T}_j, \overline{L}_j, \overline{T}_j^{\text{drive,int}}, \overline{n}_j^{\text{sb}}, \overline{T}_j^{\text{drive,day}}, \overline{LT}_j, \overline{XT}_j, \overline{D}_j)$ that is obtained using resource extension functions specific to this option. Next, we describe these resource extension functions for each option and we also specify under which conditions an option belongs to $\Delta(E_j, j)$.

Short break: Inserting a 15-minute short break along arc (i, j) only makes sense when there is unavoidable waiting at vertex j that does not last enough to insert a 45-minute long break ($0 < a_j - XT_j < 45$). Furthermore, it is feasible only if $n_j^{\text{sb}} = 0$. Finally, it is not dominated by the insertion of a long break if the maximum interval driving time is not exceeded ($T_j^{\text{drive,int}} < 270$), or by the insertion of a rest if both the maximum daily driving time and the maximum daily duration are not exceeded ($T_j^{\text{drive,day}} < 540$ and $D_j < 780$). Thus, $sb \in \Delta(E_j, j)$ if all these conditions are satisfied. Such a break can start at any time between the end of the last break or rest on the arc (i, j) and the beginning of the unavoidable waiting period. When selecting this option, the resource vector E_j is updated using the following resource extension functions to create a new resource vector $\overline{E}_j$.

$$\overline{T}_j = T_j + 15 \tag{5.13}$$

$$\overline{n}_j^{\text{sb}} = 1 \tag{5.14}$$

$$\overline{XT}_j = XT_j + 15 \tag{5.15}$$

$$\overline{LT}_j = \max\{\min\{\overline{XT}_j, b_j\}, a_j\} \tag{5.16}$$

$$\overline{D}_j = D_j + \max\{15 - (a_j - XT_j), 0\} \tag{5.17}$$

$$\overline{L}_j = L_j, \quad \overline{T}_j^{\text{drive,int}} = T_j^{\text{drive,int}}, \quad \overline{T}_j^{\text{drive,day}} = T_j^{\text{drive,day}} \tag{5.18}$$

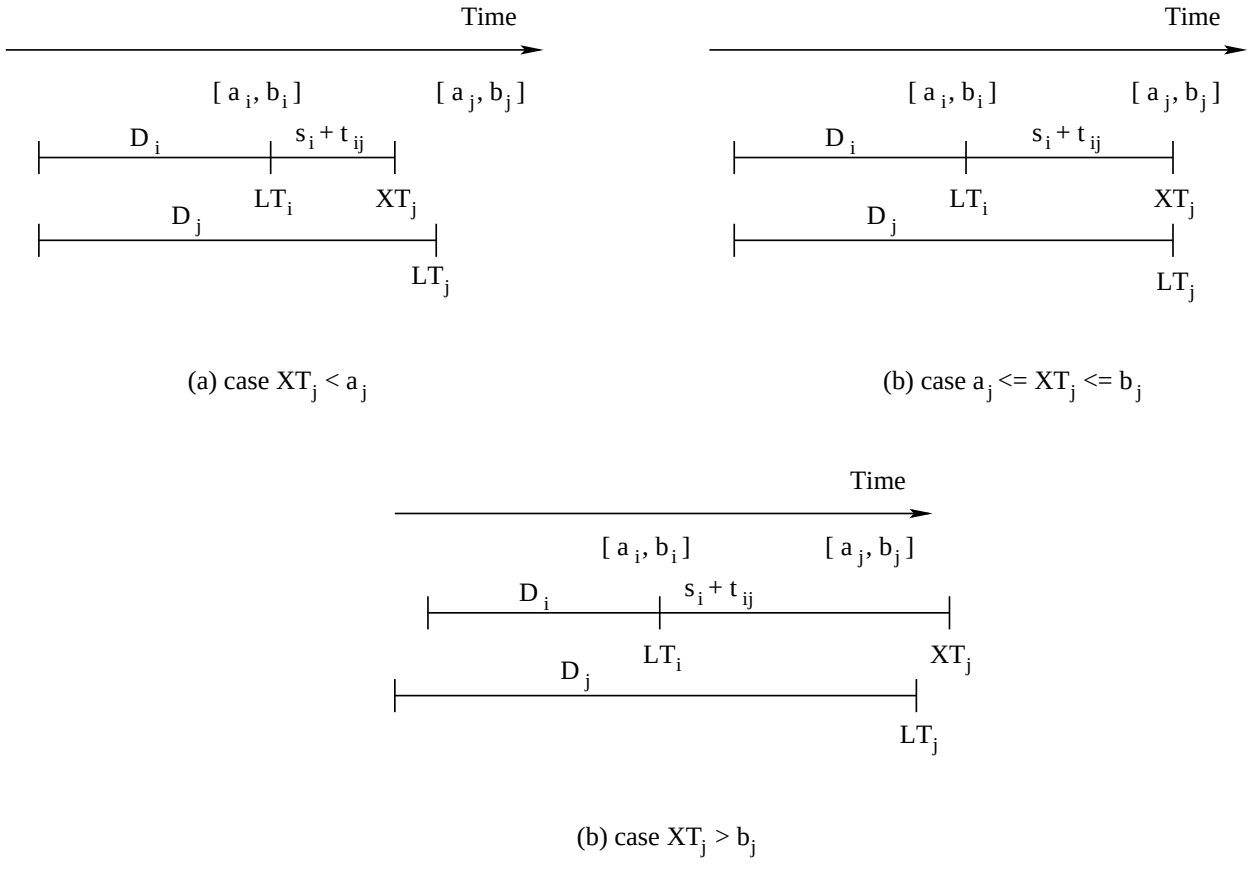

Figure 5.2 Three cases for the computation of the current day duration

We denote these resource extension functions by $REF_{sb}(\cdot,\cdot)$, that is, $\overline{E}_j = REF_{sb}(E_j, j)$. Note that breaks and rests are applied as resource component modifications. Therefore, there is no trace of the breaks and rests used unless they are kept in a dedicated memory structure.

Long break: A long break can be inserted along arc (i, j) if the maximum interval driving time is exceeded or there is unavoidable waiting time (otherwise, the insertion can always be postponed to the next arc). It is, however, dominated by a rest if the maximum daily driving time or the maximum daily duration is reached before the maximum interval driving time while driving. Hence, $lb \in \Delta(E_j, j)$ if $\max\{T_j^{\text{drive,int}}, a_j - XT_j\} > 0$ and $T_j^{\text{drive,int}} - 270 > \max\{T_j^{\text{drive,day}} - 540, D_j - \max\{a_j - XT_j, 0\} - 780\}$. If $n_j^{\text{sb}} = 1$, the duration k_j of the break is 30 minutes, otherwise $k_j = 45$ minutes. Such a break starts at the first point in time when the interval driving time reaches 270 minutes or when the unavoidable waiting period starts.

The resource extension functions for the insertion of a long break are as follows.

$$\begin{aligned}
\overline{T}_j &= T_j + k_j && (5.19)\\
\overline{T}_j^{\text{drive,int}} &= \max\{T_j^{\text{drive,int}} - 270, 0\} && (5.20)\\
\overline{n}_j^{\text{sb}} &= 0 && (5.21)\\
\overline{XT}_j &= XT_j + k_j && (5.22)\\
\overline{LT}_j &= \max\{\min\{\overline{XT}_j, b_j\}, a_j\} && (5.23)\\
\overline{D}_j &= D_j + \max\{k_j - \max\{a_j - XT_j, 0\}, 0\} && (5.24)\\
\overline{L}_j &= L_j, \quad \overline{T}_j^{\text{drive,day}} = T_j^{\text{drive,day}} && (5.25)
\end{aligned}$$

We denote these resource extension functions by $REF_{lb}(\cdot,\cdot)$, that is, $\overline{E}_j = REF_{lb}(E_j, j)$.

Long rest: A 660-minute long rest can be inserted at the time when the maximum interval driving time, the maximum daily driving time, or the maximum daily duration exceeds. It can also be inserted as the last activity along arc (i, j) if the maximum daily duration is to be reached during the service at vertex j or during an unavoidable waiting period at vertex j. If taken, the long rest starts at the earliest of these times, denoted h. There is only one restriction on the insertion of a long rest along arc (i, j): it cannot occur during the service at vertex i. Therefore, $lr \in \Delta(E_j, j)$ if h does not fall during the service at vertex i. (Note that additional notation is needed to express this condition formally. This notation has been omitted for reasons of conciseness and clarity.) The resource extension functions for the insertion of a long rest are as follows.

$$\begin{aligned}
\overline{T}_j &= T_j + 660 && (5.26)\\
\overline{L}_j &= L_j && (5.27)\\
\overline{T}_j^{\text{drive,int}} &= \max\{T_j^{\text{drive,int}} - 270, T_j^{\text{drive,day}} - 540, D_j - \max\{a_j - XT_j, 0\} - 780, 0\} && (5.28)\\
\overline{n}_j^{\text{sb}} &= 0 && (5.29)\\
\overline{T}_j^{\text{drive,day}} &= \overline{T}_j^{\text{drive,int}} && (5.30)\\
\overline{XT}_j &= XT_j + 660 && (5.31)\\
\overline{LT}_j &= b_j && (5.32)\\
\overline{D}_j &= \overline{T}_j^{\text{drive,int}} && (5.33)
\end{aligned}$$

In the maximum function of (5.28), the first three terms indicate the driving time left after the rest if it was taken at the time when the maximum interval driving time, the maximum daily driving time, or the maximum daily duration (offset by the unavoidable waiting time, if any) exceeds, respectively. When at least one of these maxima is reached before the end

of driving, the rest is taken as soon as one of them is reached, yielding the largest driving time left after the rest. This remaining driving time is then scheduled on the day starting after the rest and is, thus, used to initialize the current daily driving time and the current daily duration components in (5.30) and (5.33), respectively. In (5.31), the extended latest start of service time is increased by the duration of a long rest and is used in subsequent extensions along arc (i, j) to compute the remaining unavoidable waiting time, if any. On the other hand, the latest start of service time $\overline{LT}_j$ is set to b_j in (5.32) as service at the first customer visited in a day can always start at the upper bound of this customer time window.

The resource extension functions (5.26)–(5.33) are denoted $REF_{lr}(\cdot, \cdot)$, that is, $\overline{E}_j = REF_{lr}(E_j, j)$.

Label extension function

The pseudo-code of the label extension function $ExtendLabel(\cdot, \cdot, \cdot)$ used in Step 6 of Algorithm 5.1 is given in Algorithm 5.2. Taking a feasible label E_i associated with vertex i as input, this function relies on the resource extension functions presented above to generate feasible labels at vertex j if some exist. This function starts by extending E_i using the resource extension functions $REF_{first}(E_i, i, j)$ to yield a resource vector E_j. This extension corresponds to adding the service at vertex i and the traveling between i and j. The time-adjusted vector $AdjTime(E_j, j)$ is then tested for feasibility in Step 3. In Step 5, E_j is tested for local extendability: a label $E_j = (T_j, L_j, T_j^{\text{drive,int}}, n_j^{\text{sb}}, T_j^{\text{drive,day}}, LT_j, XT_j, D_j)$ is said to be extendable if $T_j \leq b_j$, $L_j \leq Q$, and $\max\{a_j - XT_j, T_j^{\text{drive,int}} - 270, T_j^{\text{drive,day}} - 540, D_j + s_j - 780\} > 0$ (that is, one of the following conditions must hold: there is unavoidable waiting at vertex j, the maximum interval driving time is exceeded, the maximum daily driving time is exceeded, the maximum daily duration is exceeded or will exceed during service at vertex j). If E_j is extendable, the recursive function $RecursiveExtension(E_j, j)$ is applied in Step 6 to insert breaks and rests along arc (i, j).

The pseudo-code of the $RecursiveExtension(E_j, j)$ function is given in Algorithm 5.3. This function extends E_j with each valid break and rest insertion option (loop starting in Step 2) to yield a resource vector $\overline{E}_j$ for each of these options. Each time-adjusted vector $AdjTime(\overline{E}_j, j)$ is then checked for feasibility (Step 4) and each vector $\overline{E}_j$ for extendability (Step 6). Each extendable vector is then extended recursively in Step 7.

To illustrate the application of the label extension function $ExtendLabel(E_i, i, j)$, consider the following example that extends a feasible label E_i along an arc $(i, j) \in \mathcal{A}_p$. In this example, $[a_i, b_i] = [1300, 1600]$, $[a_j, b_j] = [1700, 2800]$, $s_i = 60$, $s_j = 90$, $t_{ij} = 300$, $q_j = 20$, and $Q = 200$. The values of the components of E_i are specified in Table 5.1, together with the values of the components of the generated resource vectors. In this table, the first and

Algorithm 5.2 Function $ExtendLabel(E_i, i, j)$

Require: A label E_i, a vertex i and a vertex j such that $E_i \in \mathcal{E}(i)$ and $(i,j) \in \mathcal{A}_p$

$\mathcal{E} \leftarrow \emptyset$
$E_j \leftarrow REF_{first}(E_i, i, j)$
if $AdjTime(E_j, j)$ is feasible at vertex j **then**
  $\mathcal{E} \leftarrow \{AdjTime(E_j, j)\}$
if E_j is extendable **then**
  $\mathcal{E} \leftarrow \mathcal{E} \cup RecursiveExtension(E_j, j)$
Return $\mathcal{E}$

Algorithm 5.3 Function $RecursiveExtension(E_j, j)$

Require: A resource vector E_j and a vertex j such that E_j is associated with j.

$\mathcal{E} \leftarrow \emptyset$
for all $o \in \Delta(E_j, j)$ **do**
  $\overline{E}_j \leftarrow REF_o(E_j, j)$
  if $AdjTime(\overline{E}_j, j)$ is feasible at vertex j **then**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{AdjTime(\overline{E}_j, j)\}$
  if $\overline{E}_j$ is extendable **then**
    $\mathcal{E} \leftarrow \mathcal{E} \cup RecursiveExtension(\overline{E}_j, j)$
Return $\mathcal{E}$

second lines provide a resource vector identifier and the resource extension functions used to derive this vector, respectively. Note that the arguments i and j in those functions have been omitted to reduce the table width. Lines 3 to 10 indicate the component values. For each generated resource vector, lines 11 to 13 specify if the corresponding time-adjusted resource vector is feasible (*TA feas.*), if the vector is extendable (*extend.*) and, if this is the case, the subset of valid options. Table 5.2 details the sequence of instructions executed in Algorithms 5.2 and 5.3 to yield vectors E_1 to E_4.

For this example, the label extension function produces two resource vectors E_3 and E_4 (which corresponds to their time-adjusted vectors) that are feasible labels at vertex j. E_3 was generated by adding a long break and a long rest along arc (i, j), whereas E_4 was obtained by adding only a long rest. Service at vertex j can start earlier if E_4 (instead of E_3) is further extended along the arc originating at j. On the other hand, a break or a rest will be needed earlier in this case.

The complexity of the exact algorithm for checking route feasibility is thus exponential in the number of possible options that can be inserted along every arc of a path. This complexity makes the application of the exact algorithm impractical in the TS column generator (see Section 5.3.3) where the algorithm would need to be invoked for every customer insertion move. This algorithm can, however, be transformed into a heuristic as follows. The set of all resource vectors (including the feasible labels and the intermediate resource vectors) generated during the exact algorithm can be seen as the vertex set of an oriented tree in which an arc (i, j) indicates that resource vector j was generated from resource vector i. The proposed heuristic algorithm also explores this tree using a depth-first search strategy but limits to a predefined maximum number M^{back} the number of backtracks that it can perform and stops as soon as a feasible placement of breaks and rests is found for the whole route.

In the following, we propose a fast approximate route feasibility check that can be used when feasibility needs to be checked for several individual customer insertions into the same route. This check relies, in part, on the heuristic version of the above labeling algorithm.

5.4.3 Approximate feasibility check for multiple insertions

At each iteration, the TS heuristic, described in Section 5.3.3, evaluates a certain number of individual customer insertions into the current solution (a feasible path $p = i_0 - i_1 - i_2 - \ldots - i_k - i_{k+1}$). Each potential insertion results in a new route that must be checked for feasibility. In this subsection, we propose an approximate checking procedure that can be used to identify rapidly certain infeasible routes obtained by a customer insertion. This strategy works in two steps. First, a preprocessing step that computes lower and upper bounds on certain resource values at each vertex of path p is performed once for this path.

	E_i	$E_1 = REF_{first}(E_i)$	$E_2 = REF_{lb}(E_1)$	$E_3 = REF_{lr}(E_2)$	$E_4 = REF_{lr}(E_1)$
T	1300	1660	1705	2365	2320
L	50	70	70	70	70
$T^{\text{drive, int}}$	150	450	180	80	180
n^{sb}	0	0	0	0	0
$T^{\text{drive, day}}$	320	620	620	80	180
XT	1580	1940	1985	2645	2600
LT	1580	1940	1985	2800	2800
D	400	760	805	80	180
TA feas.		no	no	yes	yes
extend.		yes	yes	no	no
options		lb, lr	lr		

Table 5.1 Example of resource vectors generated by the label extension function

Then, in a second step, an inexpensive check is executed for each customer insertion to verify if these bounds can be respected by this insertion.

Resource bound tightening

This preprocessing step relies on the assumption that a low resource value in a label at a given vertex always offers more flexibility than a higher resource value for further extensions. This is true for instance for the load resource. Indeed, to respect the vehicle capacity (that is, the load resource upper bound at every vertex), a low resource value is preferable over a higher resource value. As stated in Irnich et Desaulniers (2005), this property holds for every resource whose resource extension functions are non-decreasing. It is, however, not the case for the current daily duration resource as its future value also depends on how much the start of the current day can be shifted backwards. It is also not the case for the earliest start of service resource whose extension functions depend on the number of short breaks. Indeed, the duration of a long break, which influences the start of service time resource, depends on the previous use or not of a short break. As we want to compute only lower and upper bounds, we can however forbid the placement of short breaks for these computations and assume that all long breaks last 30 minutes. With these assumptions, the (modified) extension functions for the start of service time resource become non-decreasing. Consequently, the procedure involves only the following resources: earliest start of service time, load, interval driving time, and daily driving time. The set of these resources is denoted $\bar{R}$.

(Algo,lines)	Description
(2,2)	Starting from the feasible label E_i, the function $ExtendLabel(E_i, i, j)$ uses the extension functions $REF_{first}(E_i, i, j)$ to generate a first resource vector E_1.
(2,3)	Because $T^{\text{drive, int}} = 450 > 270$ in E_1, $AdjTime(E_1, j)$ is not feasible and, thus, not added to $\mathcal{E}$.
(2,5)	E_1 is extendable because $T = 1660 \leq 2800$, $L = 70 \leq 200$ and $\max\{a_j - XT, T^{\text{drive,int}} - 270, T^{\text{drive,day}} - 540, D + s_j - 780\} = 180 > 0$. Options lb and lr are possible because $T^{\text{drive,int}} > 270$ and $T^{\text{drive,day}} > 540$.
(2,6)	For extending E_1, function $RecursiveExtension(E_1, j)$ is called.
(3,3)	Vector E_1 is extended for option lb using $REF_{lb}(E_1, j)$ yielding vector E_2.
(3,4)	Because $T^{\text{drive, day}} = 620 > 540$ in E_2, $AdjTime(E_2, j)$ is not feasible.
(3,6)	E_2 is extendable because $T = 1705 \leq 2800$, $L = 70 \leq 200$ and $\max\{a_j - XT, T^{\text{drive,int}} - 270, T^{\text{drive,day}} - 540, D + s_j - 780\} = 80 > 0$. Option lr is possible because $T^{\text{drive,day}} > 540$.
(3,7)	Function $RecursiveExtension(E_2, j)$ is called for extending E_2.
(3,3)	Vector E_2 is extended for option lr using $REF_{lr}(E_2, j)$ yielding vector E_3.
(3,4-5)	Label $AdjTime(E_3, j)$ is feasible and added to $\mathcal{E}$.
(3,6)	Because $\max\{a_j - XT, T^{\text{drive,int}} - 270, T^{\text{drive,day}} - 540, D + s_j - 780\} \not> 0$, E_3 is not extendable. The function returns $\mathcal{E}$ (no more options for E_2).
(3,3)	Vector E_1 is extended for option lr using $REF_{lr}(E_1, j)$ yielding vector E_4.
(3,4-5)	Label $AdjTime(E_4, j)$ is feasible and added to $\mathcal{E}$.
(3,6)	Because $\max\{a_j - XT, T^{\text{drive,int}} - 270, T^{\text{drive,day}} - 540, D + s_j - 780\} \not> 0$, E_4 is not extendable. The function returns $\mathcal{E}$ (no more options for E_1).

Table 5.2 Application of Algorithms 5.2 and 5.3 yielding the resource vectors of Table 5.1

Let $lb_{i_\ell}(r)$ and $ub_{i_\ell}(r)$ be the lower and upper bounds computed for resource $r \in \bar{R}$ at vertex i_ℓ, $\ell \in \{0, 1, \ldots, k+1\}$. To compute these bounds, we use two labeling algorithms where a label E contains one component for each resource $r \in \bar{R}$, denoted $E(r)$. The first algorithm is a forward labeling algorithm that starts from a label at vertex i_0 and uses the resource extension functions presented in Section 5.4.2 (except that the duration of all long breaks is 30 minutes as mentioned above) and the label extension function $ExtendLabel(\cdot, \cdot, \cdot)$ (see Algorithm 5.2) without the short break option to create forward labels at vertices i_ℓ, $\ell \in \{1, 2, \ldots, k+1\}$. The second algorithm is a backward labeling algorithm that starts from a label at vertex i_{k+1} and uses backward resource extension functions and a backward label extension function to compute backward labels at vertices i_ℓ, $\ell \in \{0, 1, 2, \ldots, k\}$. Such reverse functions can easily be determined (see Irnich, 2008). For instance, for the earliest start of service time resource and the load resource, the backward extension functions along

an arc $(i, j) \in \mathcal{A}_p$ (the counterparts of (5.5)–(5.6)) are:

$$T_i = T_j - s_i - t_{ij} \tag{5.34}$$

$$L_i = L_j - q_i \tag{5.35}$$

with $T_{i_{k+1}} = b_{i_{k+1}} = H$ and $L_{i_{k+1}} = Q$. For the sake of conciseness, all the other backward resource extension functions are not stated here. The sets of forward and backward labels generated at a vertex i_ℓ, $\ell \in \{0, 1, 2, \ldots, k+1\}$, are denoted $\mathcal{E}^{fw}_{i_\ell}$ and $\mathcal{E}^{bw}_{i_\ell}$, respectively.

The pseudo-code of the preprocessing algorithm is given in Algorithm 5.4. In Step 1, only the lower bounds $lb_{i_0}(r)$ at the source vertex i_0 of path p are initialized. No initial lower bounds are required for the subsequent vertices along p. Step 2 initializes all the upper bounds $ub_{i_\ell}(r)$. In Steps 5–7, the forward labeling algorithm is used to compute resource value lower bounds. At each vertex i_ℓ, a single lower bound label lb_{i_ℓ} is obtained by first extending forwardly in Step 6 the single label $lb_{i_{\ell-1}}$ to generate the forward label set $\mathcal{E}^{fw}_{i_\ell}$ and then computing in Step 7 the minimal value $E(r)$ over all labels $E \in \mathcal{E}^{fw}_{i_\ell}$ for each resource $r \in \bar{R}$. Symmetrically, backward labeling and upper bound computation are performed in Steps 8–10. Note that not all feasible labels E are accepted in Steps 6 and 9. To be accepted, they must respect the conditions $E(r) \leq ub_{i_\ell}(r), \forall r \in \bar{R}$ for a forward label and the conditions $E(r) \geq lb_{i_\ell}(r), \forall\, r \in \bar{R}$ for a backward label. These conditions ensure that the computed label can be feasibly extended up to vertex i_{k+1} or vertex i_0 along path p. For instance, assume that, at a given vertex $i_{\bar{\ell}}$ along p, $[a^T_{i_{\bar{\ell}}}, b^T_{i_{\bar{\ell}}}] = [1800, 2100]$ and $ub_{i_{\bar{\ell}}}(T) = 2000$, where $r = T$ denotes the earliest start of service time resource. The upper bound $ub_{i_{\bar{\ell}}}(T)$ indicates that, to reach the sink vertex i_{k+1} from vertex $i_{\bar{\ell}}$ along p, the service at vertex $i_{\bar{\ell}}$ cannot start later than 2000. Now, the forward labeling algorithm can generate a feasible label E with $E(T) = 2050$. Even if it is feasible, this label cannot yield a feasible path up to i_{k+1} and it would not be included in set $\mathcal{E}^{fw}_{i_{\bar{\ell}}}$. Because the upper bounds are initially set to the resource window upper bounds in Step 2, the first execution of the backward labeling algorithm in Steps 9–10 can often reduce these upper bounds. In this case, it can be advantageous to recompute the lower bounds taking into account these updated upper bounds. In fact, as soon as one set of bounds change, the other set can be recomputed to yield tighter bounds. Consequently, Steps 4–10 are embedded into a repeat loop that stops either when none of the lower and upper bounds changed in the last iteration or when a maximum number of iterations is reached. Preliminary tests showed that a maximum of two iterations is sufficient to obtain good-quality bounds. Note that each iteration starts by emptying the sets $\mathcal{E}^{fw}_{i_\ell}$ and $\mathcal{E}^{bw}_{i_\ell}$ (Step 4).

To illustrate this preprocessing step, consider the example presented at the end of Sec-

Algorithm 5.4 Computing lower and upper bounds for resource values along a given route

Require: Route (path) $p = i_0 - i_1 - i_2 - \ldots - i_k - i_{k+1}$
1: $lb_{i_0}(r) \leftarrow a^r_{i_0}, \forall\, r \in \bar{R}$
2: $ub_{i_\ell}(r) \leftarrow b^r_{i_\ell}, \forall\, \ell \in \{0, 1, \ldots, k+1\}, r \in \bar{R}$
3: **repeat**
4: $\quad \mathcal{E}^{fw}_{i_\ell} \leftarrow \emptyset$, $\mathcal{E}^{bw}_{i_\ell} \leftarrow \emptyset$, $\forall\, \ell \in \{0, 1, \ldots, k+1\}$
5: $\quad$ **for** $\ell = 1, 2, \ldots, k+1$ **do**
6: $\quad\quad$ Compute $\mathcal{E}^{fw}_{i_\ell}$, the set of all feasible labels E obtained by extending forwardly label $lb_{i_{\ell-1}}$ along arc $(i_{\ell-1}, i_\ell)$ and such that $E(r) \leq ub_{i_\ell}(r), \forall\, r \in \bar{R}$.
7: $\quad\quad lb_{i_\ell}(r) \leftarrow \min_{E \in \mathcal{E}^{fw}_{i_\ell}} E(r), \forall\, r \in \bar{R}$
8: $\quad$ **for** $\ell = k, k-1, \ldots, 0$ **do**
9: $\quad\quad$ Compute $\mathcal{E}^{bw}_{i_\ell}$, the set of all feasible labels E obtained by extending backwardly label $ub_{i_{\ell+1}}$ along arc $(i_\ell, i_{\ell+1})$ and such that $E(r) \geq lb_{i_\ell}(r), \forall\, r \in \bar{R}$.
10: $\quad\quad ub_{i_\ell}(r) \leftarrow \max_{E \in \mathcal{E}^{bw}_{i_\ell}} E(r), \forall\, r \in \bar{R}$
11: **until** some stop criterion is met

tion 5.4.2 (discarding all resources not in $\bar{R}$). Assume that, at beginning of an iteration of the repeat loop, $lb_i = (1300, 50, 150, 320)$ (that is, $lb_i = E_i$ in Table 5.1) and $ub_j = (2650, 100, 150, 220)$, where the components in these vectors correspond to components T, L, $T^{\text{drive, int}}$, and $T^{\text{drive, day}}$ of the resource vectors. By extending label lb_i along arc (i, j) in Step 6, only two feasible labels at vertex j are created, namely, $E_3 = (2365, 70, 80, 80)$ and $E_4 = (2320, 70, 180, 180)$. Then, both labels respect the upper bound conditions and $\mathcal{E}^{fw}_j = \{E_3, E_4\}$. Computing the lowest value over all labels in $\mathcal{E}^{fw}_j$ for each resource in Step 7, we obtain $lb_j = (2320, 70, 80, 80)$. Afterwards, the forward labeling algorithm extends this new label until reaching vertex i_{k+1} before starting the backward labeling algorithm in Step 9. Now, assume that this latter algorithm computes the following three backward labels when extending the label ub_g along arc (j, g): $E_5 = (2250, 100, 50, 200)$, $E_6 = (2575, 100, 140, 140)$, and $E_7 = (2450, 100, 120, 160)$. Because $E_5(T) = 2250 < lb_j(T) = 2320$, E_5 does not respect the lower bound conditions. Thus, $\mathcal{E}^{bw}_j = \{E_6, E_7\}$ and ub_j can be updated to $ub_j = (2575, 100, 140, 160)$ in Step 10. If there were a next iteration in the repeat loop and both labels E_3 and E_4 were generated again in Step 6, then E_4 would not belong to $\mathcal{E}^{fw}_j$ because it would violate the updated upper bound conditions. This would also yield an updated lower bound vector lb_j. Note that there will always be at least one label in each set $\mathcal{E}^{fw}_j$ and $\mathcal{E}^{bw}_j$. Otherwise, path p would not be feasible.

Filtering infeasible insertions

Once the lower and upper bounds are computed for path p, the route feasibility check for each customer insertion can be performed. If customer j is to be inserted between customers i_ℓ and $i_{\ell+1}$ in p, then we extend label lb_{i_ℓ} along the arcs (i_ℓ, j) and $(j, i_{\ell+1})$ using the resource and label extension functions of Sections 5.4.2 and 5.4.2. If there are no feasible labels E at vertex $i_{\ell+1}$ that satisfy $E(r) \leq ub_{i_{\ell+1}}(r)$, $\forall\, r \in \bar{R}$, then the insertion is infeasible. The complexity of this check is thus independent of the route length, except for the insertion of an aggregated customer (sequence of customers fixed for the current LNS iteration) that requires extending label lb_{i_ℓ} through the whole arc sequence.

Furthermore, in the present context of column generation, the reduced cost $rc(i)$ of a route created by a customer insertion i can easily be computed in constant time. When multiple (say k) insertions need to be checked, one can retain the reduced cost rc_{min} of the best feasible insertion tested so far. After evaluating insertion $j \in \{1, 2, \ldots, k-1\}$, this minimal reduced cost is $rc_{min} = \min_{i \in I_j} rc(i)$, where $I_j \subseteq \{1, 2, \ldots, j\}$ is the subset of feasible insertions already tested. Then, if $I_j \neq \emptyset$ and $rc(j+1) \geq rc_{min}$, it is not necessary to check the feasibility of insertion $j+1$ because, even if it was feasible, it would be dominated by the cheapest insertion found so far.

Identifying feasible insertions

Every customer insertion that has not been filtered out (proven infeasible) by the previous procedure needs to be checked for feasibility. This check is performed using the heuristic version of the labeling algorithm exposed in Subsections 5.4.1 and 5.4.2. Note that, during this search, the bounds lb_j and ub_j computed in the preprocessing step of the bound tigthening procedure (Algorithm 5.4) are used to restrict the resource values at each vertex i, instead of the original resource windows $[a_i^r, b_i^r]$ for all $i \in \mathcal{N}_p$ and $r \in R$.

If this heuristic check cannot prove the feasibility of the resulting route, then the insertion is considered infeasible (even though it could be feasible) and rejected in the TS column generator. Unfortunately, rejecting feasible insertions reduces the possibility of finding certain negative reduced cost columns with the heuristic TS column generator.

5.5 Computational experiments

This section presents some computational results on the instances proposed by Goel (2009). These instances are derived from the well-known benchmark VRPTW instances of Solomon (1987) that are grouped into 6 classes: R1, C1, RC1, R2, C2, and RC2. In the R1 and R2 classes, the customers are randomly distributed in a square region. They are

clustered in the C1 and C2 classes. The customer distribution is mixed in the RC1 and RC2 classes. The R2, C2, and RC2 have larger time windows than the R1, C1, and RC1 instances, increasing considerably the number of feasible routes. In total, there are 56 instances (between 8 and 12 per instance class) which all involve 100 customers. These VRPTW instances are modified as follows for the VRPTWDR. The time windows are multiplied proportionally to obtain a time horizon of 144 hours and the traveling speed is set at 5 units of distance per hour, instead of 60 as in the initial Solomon's instances. Service time at every customer is fixed to 60 minutes. With these modifications, certain customers in certain instances cannot be visited without violating their time windows because of the additional time required for breaks and rests in the transit between the depot to the customer or between the customer to the depot. Consequently, to yield feasible instances, the time windows of these customers are further modified to ensure that they can always be reached directly from the depot and that the depot can be reached after visiting them (see Goel, 2009).

For all experiments with the proposed LNS algorithm, the values of the parameters (see Section 5.3.1) were as follows: $I_1^{max} = 100$ (maximum number of iterations to reduce the number of vehicles used by one), $I_2^{max} = 100$ (maximum number of iterations to reduce the total distance), and $M^{rem} = 60$ (number of customers to remove). Furthermore, the heuristic route feasibility check algorithm of Sections 5.4.1 and 5.4.2 was limited to $M^{back} = 5$ backtracks per customer insertion. All our experiments were conducted on an AMD Opteron processor clocked at 2.3GHz.

As in Kok *et al.* (2010), we performed three sets of experiments that differ by the driver rules considered. The first tests aimed at evaluating the performance of our algorithm on the necessary set of rules that enforce feasibility with respect to Regulation (EC) No 561/2006 alone: namely, a driver must take a 45-minute break after 4.5 hours of driving, he must take an 11-hour rest after 9 hours of driving, the weekly driving time is limited to 56 hours, and a daily rest must be taken within 24 hours after the end of the last daily rest. It is thus a set of strict rules that do not allow exceptions nor break/rest splitting. Table 5.3 summarizes the computed results (columns PDDR) by instance class and compares them with the results obtained by Goel (2009) andKok *et al.* (2010) (column KMKS). For each instance class, the first line indicates the average number of vehicles used over all the instances in the class, whereas the second line provides the average total distance. For our algorithm and that of Goel (2009) which both include randomness, five runs were performed for each instance. The *Best* columns give the best results over the five runs, whereas the *Avg* columns provide the average results. The lines CNV and CTD present the cumulative number of vehicles used and the cumulative total distance over all instances, respectively. Finally, the processor used (CPU: Pentium 4 for Goel, Pentium M for Kok *et al.*) and the average computational time

per instance and run (in minutes) are given at the bottom of the table, together with the number of runs per instance.

For these first experiments, our LNS algorithm clearly outperforms the algorithms of Goel (2009) and Kok *et al.* (2010). Indeed, the solutions computed by our algorithm require much less vehicles (the solutions of Goel and Kok *et al.* use, respectively, 46% and 21% more vehicles than our solutions), which is the primary objective of the VRPTWDR. Furthermore, the cumulative total distance is also reduced significantly. Finally, notice that our LNS algorithm is faster than the algorithm of Goel (2009), but much slower than the algorithm of Kok *et al.* (2010).

In addition to the first set of rules, we then considered the working time rules of Directive 2002/15/EC, which specify that the working time (composed of driving and servicing time) may not exceed 6 hours consecutively and 60 hours weekly. Considering this augmented set of rules now yields feasible solutions with respect to all regulations, since only exceptions and relaxations that provide more flexibility are omitted. Table 5.4 compares the results of our LNS algorithm for this second set of rules against those obtained by Kok *et al.* (2010), the only authors taking working time regulations into account. Again, our results are much better than those of Kok *et al.* (2010). As expected, considering additional constraints yields solutions requiring more vehicles. However, the total distance traveled is more or less the same. Finally, observe that the treatment of these additional rules has almost no impact on the computational times.

The third set of experiments involved the complete set of rules, including the possibility to use short breaks, short rests, and reduced rests, and to extend the maximum daily driving time. Table 5.5 reports the results for these experiments. Again, we clearly see the superiority of the proposed LNS algorithm over the algorithm of Kok *et al.* (2010): compared to our solutions, the solutions of Kok *et al.* require 16% more vehicles and produce an increase of 15% in the total distance traveled. However, it is important to notice that Kok *et al.* (2010) approach is extremely fast as it computes solutions within only one minute of computation time even when all the rules are considered, as opposed to an average of almost 90 minutes for our algorithm.

Finally, we performed a last series of tests to assess the quality of the solutions produced by the proposed LNS algorithm for reduced computational times. Still considering the complete set of rules, we achieved faster computational times by reducing the size of the neighborhoods explored at each LNS iteration (that is, by reducing the value of M^{rem}, the number of customers removed) and keeping the same values for the other parameters. Table 5.6 provides the results of these experiments for values of M^{rem} varying between 60 (the value used in the previous tests) and 10. These results present averages over five runs. As expected, reducing

	PDDR		Goel		KMKS
	Best	Avg	Best	Avg	
C1	10.00	10.00	11.11	12.04	10.33
	847.69	847.70	1054.45	1096.58	965.44
C2	4.38	4.38	8.38	9.58	5.00
	688.64	694.47	954.64	1008.71	770.42
R1	8.08	8.13	10.92	11.93	9.67
	997.18	993.91	1144.23	1180.96	1152.39
R2	5.00	5.04	10.27	11.27	7.55
	948.51	957.64	1107.14	1151.24	1100.83
RC1	9.00	9.00	11.13	12.10	10.25
	1112.51	1117.82	1347.75	1373.10	1300.60
RC2	5.88	5.93	10.00	11.43	8.13
	1127.75	1127.77	1347.26	1389.50	1266.64
CNV	396	397.4	580	640.4	479
CTD	53460	53611	64596	66875	61328
CPU	OPT 2.3Ghz	OPT 2.3Ghz	P4 2.8GHz	P4 2.8GHz	PM 2.0GHz
Time (min)		10		30	1
Runs		5		5	1

Table 5.3 Results for the necessary set of rules of Regulation (EC) No 561/2006

the size of the neighborhoods (without changing the number of LNS iterations) accelerates the average computational time, but deteriorates solution quality. However, we can see that our algorithm still outperforms the algorithm of Kok *et al.* (2010) for similar computational times.

5.6 Conclusion

In this paper, we proposed a large neighborhood search algorithm based on a column generation heuristic to solve the VRPTWDR. The column generation heuristic relies on a tabu search algorithm for generating routes dynamically, as well as a heuristic labeling algorithm for checking the feasibility of the routes. The proposed solution method, which extends the previous work of Prescott-Gagnon *et al.* (2009), yields state-of-the-art results for known VRPTWDR benchmark instances.

Although it has been shown in Archetti et Savelsbergh (2009) that it is possible to construct, in polynomial time, a feasible driver schedule for a given route under the legislation of the United States, it remains unknown if this problem is polynomial or non-polynomial when the European regulations are considered. Identifying this complexity would constitute an interesting contribution for the VRPTWDR and could lead to improved solution methods.

Acknowledgements: Thanks to Christoph Rang for very helpful advice on the legal aspects of driver rules. This research was funded by the Bundesministerium für Wirtschaft

	PDDR		KMKS
	Best	Avg	
C1	10.00	10.00	10.33
	847.61	847.63	949.31
C2	5.00	5.00	5.75
	724.08	730.88	834.37
R1	8.17	8.22	9.67
	987.95	987.94	1155.89
R2	5.73	5.73	7.91
	932.95	940.17	1097.26
RC1	9.00	9.00	10.25
	1112.93	1118.23	1300.14
RC2	6.25	6.40	8.50
	1122.04	1117.37	1264.52
CNV	413	414.8	492
CTD	53419	53558	61677
CPU	OPT 2.3Ghz	OPT 2.3Ghz	PM 2.0GHz
Time (min)		11	1
Runs		5	1

Table 5.4 Results for the necessary set of rules of Regulation (EC) No 561/2006 and Directive 2002/15/EC

und Technologie (German Federal Ministry of Economics and Technology) under grant no. 19G7032A and by a team research grant from the Fonds québécois de la recherche sur la nature et les technologies.

5.7 Appendix A: Additional resources and extension functions

In Section 5.4, we presented a procedure for checking the feasibility of a route. To simplify its presentation, we did not consider all driver rules and options. In this appendix, we present the additional resources and extension functions required to handle all rules and options in this procedure.

Besides the eight resource components introduced in Section 5.4.1, a resource vector E_i at a vertex $i \in \mathcal{N}_p$ of a route p contains the following seven additional resource components (listed with their corresponding resource windows).

	PDDR		KMKS
	Best	Avg	
C1	10.00	10.00	10.11
	847.61	847.62	937.08
C2	4.63	4.63	5.25
	689.95	694.95	773.80
R1	8.08	8.08	9.33
	971.64	975.91	1142.62
R2	5.45	5.67	7.36
	933.93	928.04	1084.70
RC1	9.00	9.00	10.00
	1107.27	1111.36	1322.41
RC2	6.13	6.15	8.13
	1090.80	1096.59	1247.37
CNV	405	407.6	471
CTD	52665	52771	60826
CPU	OPT 2.3Ghz	OPT 2.3Ghz	PM 2.0GHz
Time (min)		88	1
Runs		5	1

Table 5.5 Results for the complete set of rules

	M^{rem}	CNV	CTD	Time (min.)
PDDR	60	407.6	52771	88
	50	409.4	52906	52
	40	411.4	53081	27
	30	415.6	53554	13
	20	421.8	54307	5
	10	435.6	57712	1
KMKS	-	471	60826	1

Table 5.6 Results for the complete set of rules for varying neighborhood sizes

$\beta_i^{\text{drive,ext}} \in [0,1]$: indicates if, at vertex i, the maximum daily driving time is extended or not to 10 hours for the current day.

$n_i^{\text{dur,ext}} \in [0,2]$: number of maximum daily driving time extensions taken up to vertex i.

$\beta_i^{\text{dur,ext}} \in [0,1]$: indicates if, at vertex i, the current maximum daily duration is extended to 15 hours.

$n_i^{\text{red}} \in [0,3]$: number of reduced daily rests taken up to vertex i.

$T_i^{\text{drive,week}} \in [0,3360]$: weekly driving time up to vertex i.

$T_i^{\text{work,int}} \in [0,360]$: interval working time up to vertex i.

$T_i^{\text{work,week}} \in [0,3600]$ weekly working time up to vertex i.

Thus, $E_i = (T_i, L_i, T_i^{\text{drive,int}}, n_i^{\text{sb}}, T_i^{\text{drive,day}}, LT_i, XT_i, D_i, \beta_i^{\text{drive,ext}}, n_i^{\text{drive,ext}}, \beta_i^{\text{dur,ext}}, n_i^{\text{red}}, T_i^{\text{drive,week}}, T_i^{\text{work,int}}, T_i^{\text{work,week}})$.

In the function $ExtendLabel(E_i, i, j)$ presented in Algorithm 5.2, the label E_i is extended along arc $(i, j) \in \mathcal{A}_p$ using first the resource extension functions $REF_{first}(E_i, i, j)$ introduced in Section 5.4.2 to produce a resource vector $E_j = (T_j, L_j, T_j^{\text{drive,int}}, n_j^{\text{sb}}, T_j^{\text{drive,day}}, LT_j, XT_j, D_j, \beta_j^{\text{drive,ext}}, n_j^{\text{drive,ext}}, \beta_j^{\text{dur,ext}}, n_j^{\text{red}}, T_j^{\text{drive,week}}, T_j^{\text{work,int}}, T_j^{\text{work,week}})$. Beside functions (5.5)–(5.12), this set of extension functions also includes the following functions.

$$\beta_j^{\text{drive,ext}} = \beta_i^{\text{drive,ext}}, \ n_j^{\text{drive,ext}} = n_i^{\text{drive,ext}}, \ \beta_j^{\text{dur,ext}} = \beta_i^{\text{dur,ext}}, \ n_j^{\text{red}} = n_i^{\text{red}} \tag{5.36}$$

$$T_j^{\text{drive,week}} = T_i^{\text{drive,week}} + t_{ij} \tag{5.37}$$

$$T_j^{\text{work,int}} = T_i^{\text{work,int}} + s_i + t_{ij} \tag{5.38}$$

$$T_j^{\text{work,week}} = T_i^{\text{work,week}} + s_i + t_{ij} \tag{5.39}$$

In Step 5 of Algorithm 5.2, E_j is tested for extendability using the following definition that takes into account the additional resources: a label E_j is said to be extendable if $T_j \leq b_j$, $L_j \leq Q$, $T_j^{\text{drive,week}} \leq 3360$, $T_j^{\text{work,week}} \leq 3600$, and $\max\{T_j^{\text{drive,int}} - 270, T_j^{\text{drive,day}} - 540, D_j + s_j - 780, T_j^{\text{work,int}} - 360\} > 0$.

The resource vector E_j might be infeasible, but when it is extendable, different options can be applied to possibly yield feasible labels. In Section 5.4.2, only three options were considered: inserting a short break (sb), a long break (lb), or a regular long rest (lr). Four additional options are available: inserting a short rest (sr), extending the maximum daily duration (ed), inserting a reduced daily rest (rr), or extending the maximum daily driving time (et). Given a resource vector E_j, the subset of options $\Delta(E_j, j)$ applicable along an arc $(i, j) \in \mathcal{A}_p$ is, thus, a subset of $\{sb, lb, lr, sr, ed, rr, et\}$. Applying such an option generates a new resource vector $\overline{E_j} = (\overline{T}_j, \overline{L}_j, \overline{T}_j^{\text{drive,int}}, \overline{n}_j^{\text{sb}}, \overline{T}_j^{\text{drive,day}}, \overline{LT}_j, \overline{XT}_j, \overline{D}_j, \overline{\beta}_j^{\text{drive,ext}}, \overline{n}_j^{\text{drive,ext}}, \overline{\beta}_j^{\text{dur,ext}}, \overline{n}_j^{\text{red}}, \overline{T}_j^{\text{drive,week}}, \overline{T}_j^{\text{work,int}}, \overline{T}_j^{\text{work,week}})$ that is obtained using resource extension functions specific to this option. Some of the extension functions for the options sb, lb and lr were described in Section 5.4.2. Below, we complete these extension functions for the resources introduced in this appendix. We also provide the extension functions for the options sr, ed, rr, and et.

Short break: Inserting a short break has no impact on the value of the additional resources. Such an option belongs to $\Delta(E_j, j)$ if the conditions stated in Section 5.4.2 hold and the additional condition $T_j^{\text{work,int}} < 360$ is also satisfied (otherwise, a long break or a rest is mandatory). The resource extension functions $REF_{sb}(E_j, j)$ for this option include

(5.13)–(5.18) and the following functions.

$$\begin{aligned} \overline{\beta}_j^{\text{drive,ext}} &= \beta_j^{\text{drive,ext}},\ \overline{n}_j^{\text{drive,ext}} = n_j^{\text{drive,ext}},\ \overline{\beta}_j^{\text{dur,ext}} = \beta^{\text{dur,ext}},\ \overline{n}_j^{\text{red}} = n_j^{\text{red}} && (5.40)\\ \overline{T}_j^{\text{drive,week}} &= T_j^{\text{drive,week}},\ \overline{T}_j^{\text{work,int}} = T_j^{\text{work,int}},\ \overline{T}_j^{\text{work,week}} = T_j^{\text{work,week}} && (5.41) \end{aligned}$$

Long break: Inserting a long break also modifies the value of the interval working time resource. The values of the other additional resources do not change. Such an option belongs to $\Delta(E_j, j)$ if $\max\{T_j^{\text{drive,int}} - 270, T_j^{\text{work,int}} - 360\} > 0$ or $0 < a_j - XT_j < 180$ and if $\max\{T_j^{\text{drive,int}} - 270, T_j^{\text{work,int}} - 360\} > \max\{T_j^{\text{drive,day}} - 540, D_j - \max\{a_j - XT_j, 0\} - 780\}$. These conditions indicate that a long break is needed if the maximum interval driving time or the maximum working time is reached, or if there is unavoidable waiting that does not exceed the minimum duration of a short rest (180 minutes). However, the insertion of a long break is dominated by the insertion of a long rest if the maximum daily drive time or the maximum daily duration is reached before reaching the maximum interval driving or working time. Note also that $lb \notin \Delta(E_j, j)$ if the break is to be inserted during service at vertex i.

The resource extension functions $REF_{lb}(E_j, j)$ for this option include (5.19)–(5.25), except function (5.20) which is replaced by

$$\overline{T}_j^{\text{drive,int}} = \max\{T_j^{\text{drive,int}} - 270, T_j^{\text{work,int}} - 360, 0\}, \tag{5.42}$$

and the following functions.

$$\begin{aligned} \overline{T}_j^{\text{work,int}} &= \overline{T}_j^{\text{drive,int}} && (5.43)\\ \overline{\beta}_j^{\text{drive,ext}} &= \beta_j^{\text{drive,ext}},\ \overline{n}_j^{\text{drive,ext}} = n_j^{\text{drive,ext}},\ \overline{\beta}_j^{\text{dur,ext}} = \beta_j^{\text{dur,ext}} && (5.44)\\ \overline{n}_j^{\text{red}} &= n_j^{\text{red}},\ \overline{T}_j^{\text{drive,week}} = T_j^{\text{drive,week}},\ \overline{T}_j^{\text{work,week}} = T_j^{\text{work,week}} && (5.45) \end{aligned}$$

Long rest: Taking a long rest resets some of the resource values. The duration of this rest is 660 minutes if $\beta_j^{\text{dur,ext}} = 0$, and 540 minutes otherwise. As before, $lr \in \Delta(E_j, j)$ if this rest does not have to be taken during service at vertex i. The resource extension functions $REF_{lr}(E_j, j)$ for this option include (5.26)–(5.33), except (5.28) which is replaced by

$$\overline{T}_j^{\text{drive,int}} = \max\{T_j^{\text{drive,int}} - 270, T_j^{\text{work,int}} - 360, T_j^{\text{drive,day}} - 540, D_j - \max\{a_j - XT_j, 0\} - 780, 0\}, \tag{5.46}$$

and the following functions.

$$\overline{\beta}_j^{\text{drive,ext}} = 0,\ \overline{\beta}_j^{\text{dur,ext}} = 0 \tag{5.47}$$
$$\overline{T}_j^{\text{work,int}} = \overline{T}_j^{\text{drive,int}} \tag{5.48}$$
$$\overline{n}_j^{\text{drive,ext}} = n_j^{\text{drive,ext}},\ \overline{n}_j^{\text{red}} = n_j^{\text{red}},\ \overline{T}_j^{\text{drive,week}} = T_j^{\text{drive,week}},\ \overline{T}_j^{\text{work,week}} = T_j^{\text{work,week}} \tag{5.49}$$

Short rest: The insertion of a short rest (option sr) is treated similarly to the insertion of a long break, the only difference being the duration of the rest that is set to 180 minutes. It belongs to $\Delta(E_j, j)$ if $\max\{T_j^{\text{drive,int}} - 270, T_j^{\text{work,int}} - 360\} > \max\{T_j^{\text{drive,day}} - 540, D_j - \max\{a_j - XT_j, 0\} - 780\}$ and $\beta_j^{\text{dur,ext}} = 0$, unless the rest has to be taken during service at vertex i. The second condition ensures that no two consecutive short rests are taken or that a short rest is taken before a reduced daily rest (in which case it is dominated). The resource extension functions $REF_{sr}(E_j, j)$ for this option include the extension functions (5.19), (5.21)–(5.25) and (5.42) with $k_j = 180$. Because the next long rest after a short rest only needs to last 9 hours, the maximum daily duration can be extended to 15 hours, which can be enforced by keeping the upper bound to 13 hours and reducing the updated daily duration by 2 hours. Also, setting $\beta_j^{\text{dur,ext}}$ to 1 allows a next long rest of 9 hours and ensures that no other short rests will be taken until then. This is adjusted by the following resource extension functions.

$$\overline{D}_j = D_j - 120 \tag{5.50}$$
$$\overline{\beta}_j^{\text{dur,ext}} = 1 \tag{5.51}$$

The rest of the resource extension functions $REF_{sr}(E_j, j)$ are as follows:

$$\overline{T}_j^{\text{work,int}} = \overline{T}_j^{\text{drive,int}} \tag{5.52}$$
$$\overline{\beta}_j^{\text{drive,ext}} = \beta_j^{\text{drive,ext}},\ \overline{n}_j^{\text{drive,ext}} = n_j^{\text{drive,ext}} \tag{5.53}$$
$$\overline{T}_j^{\text{drive,week}} = T_j^{\text{drive,week}},\ \overline{T}_j^{\text{work,week}} = T_j^{\text{work,week}}. \tag{5.54}$$

Extended maximum daily duration: Extending the maximum daily duration from 13 hours to 15 hours (option ed) induces a reduction of the duration of the next long rest (from 11 to 9 hours). It belongs to $\Delta(E_j, j)$ if $D_j \geq 780$, $D_j - \max\{a_j - XT_j, 0\} - 780 > \max\{T_j^{\text{work,int}} - 360, T_j^{\text{drive,int}} - 270, T_j^{\text{drive,day}} - 540\}$, $\beta_j^{\text{dur,ext}} = 0$, and $n_j^{\text{red}} < 3$. This means that this option can be taken if the maximum daily duration is exceeded and this occurs before exceeding the maximum interval driving time, the maximum daily driving time, and the maximum interval working time. Furthermore, if the maximum daily duration is already extended ($\beta_j^{\text{dur,ext}} = 1$), it cannot be extended further. Note that this condition ($\beta_j^{\text{dur,ext}} = 1$)

is also set when taking a short rest, which can be shown to be dominated by the extension of the maximum daily duration. Finally, option *ed* is not allowed if three reduced daily rests were already taken during the week.

For this option, the resource extension functions $REF_{ed}(E_j, j)$ are as follows.

$$\begin{aligned}
\overline{D}_j &= D_j - 120 && (5.55)\\
\overline{\beta}_j^{\text{dur,ext}} &= 1 && (5.56)\\
\overline{n}_j^{\text{red}} &= n_j^{\text{red}} + 1 && (5.57)\\
\overline{\beta}_j^{\text{drive,ext}} &= \beta_j^{\text{drive,ext}},\ \overline{n}_j^{\text{drive,ext}} = n_j^{\text{drive,ext}},\ \overline{T}_j^{\text{drive,week}} = T_j^{\text{drive,week}} && (5.58)\\
\overline{T}_j^{\text{work,int}} &= T_j^{\text{work,int}},\ \overline{T}_j^{\text{work,week}} = T_j^{\text{work,week}},\ \overline{T}_j = T_j,\ \overline{L}_j = L_j && (5.59)\\
\overline{T}_j^{\text{drive,int}} &= T_j^{\text{drive,int}},\ \overline{T}_j^{\text{drive,day}} = T_j^{\text{drive,day}},\ \overline{n}_j^{\text{sb}} = n_j^{\text{sb}}\ \overline{XT}_j = XT_j,\ \overline{LT}_j = LT_j && (5.60)
\end{aligned}$$

Reduced daily rest: As mentioned above, a long rest can be of reduced duration if $\beta_j^{\text{drive,ext}} = 1$, that is, if a short rest was taken before or if the maximum daily duration was extended. Beside these cases, a reduced daily rest of 540 minutes can also be inserted instead of a 660-minute long rest just to save time and be able to reach a customer before the end of its time window. In this case, such an insertion is treated similarly to the insertion of a long rest. Therefore, option *rr* belongs to $\Delta(E_j, j)$ if $\beta_j^{\text{dur,ext}} = 0$ and $n_j^{\text{red}} < 3$, unless it is to be scheduled during service at vertex i.

The resource extension functions $REF_{rr}(E_j, j)$ are the same as those of option *lr*, except that the rest always lasts 540 minutes and the component counting the number of reduced daily rests is updated as follows.

$$\overline{n}_j^{\text{red}} = n_j^{\text{red}} + 1 \tag{5.61}$$

Extended maximum daily driving time: Extending the maximum daily driving time from 9 to 10 hours (option *et*) belongs to $\Delta(E_j, j)$ if $T_j^{\text{drive,day}} \geq 540$, $\beta_j^{\text{drive,ext}} = 0$, $n_j^{\text{drive,ext}} < 2$ and $T_j^{\text{drive,day}} - 540 > \max\{T_j^{\text{work,int}} - 360, T_j^{\text{drive,int}} - 270, D_j - \max\{a_j - XT_j, 0\} - 780\}$. These conditions state that the maximum daily driving time must be exceeded, this maximum has not been extended for the current day, this maximum has not been extended twice during the week, and the maximum daily driving time must exceed before the other resources. The daily driving time resource is reduced by one hour in order to extend the maximum daily driving time from 9 to 10 hours, resulting in the following resource extension functions:

$$\begin{aligned}
\overline{T}_j^{\text{drive,day}} &= T_j^{\text{drive,day}} - 60 && (5.62)\\
\overline{\beta}_j^{\text{drive,ext}} &= 1 && (5.63)\\
\overline{n}_j^{\text{drive,ext}} &= n_j^{\text{drive,ext}} + 1. && (5.64)
\end{aligned}$$

The other resource values are not affected by this option. Thus, beside functions (5.62)–(5.64), the resource extension functions $REF_{ed}(E_j, j)$ include the following functions.

$$\overline{T}_j = T_j,\ \overline{L}_j = L_j,\ \overline{T}_j^{\text{drive,int}} = T_j^{\text{drive,int}},\ \overline{n}_j^{\text{sb}} = n_j^{\text{sb}} \tag{5.65}$$

$$\overline{XT}_j = XT_j,\ \overline{LT}_j = LT_j,\ \overline{D}_j = D_j,\ \overline{T}_j^{\text{work,int}} = T_j^{\text{work,int}} \tag{5.66}$$

$$\overline{T}_j^{\text{work,week}} = T_j^{\text{work,week}},\ \overline{n}_j^{\text{red}} = n_j^{\text{red}},\ \overline{T}_j^{\text{drive,week}} = T_j^{\text{drive,week}} \tag{5.67}$$

The introduction of the additional resources has also an impact on the resource bound tightening procedure described in Subsection 5.4.3. Indeed, when computing the lower and upper bounds $lb_{i_l}(r)$ and $ub_{i_l}(r)$ for the resources $r \in \bar{R}$ at all vertices i_ℓ of a path $p = i_0 - i_1 - \ldots - i_k - i_{k+1}$, neither short breaks, nor short rests are considered. Instead, the duration of all long breaks is set to 30 minutes and that of all long rests to 9 hours. Moreover, the following relaxation is also used: the value of the daily driving time resource is set to -60 at the beginning of each day to take into account the extended 10-hour maximum daily driving time. These relaxations do not necessarily yield the tightest bounds but ensure the validity of the computed bounds. Considering them, the set $\bar{R}$ of resources involved in the resource bound tightening algorithm can be augmented by the weekly driving time, the interval working time, and the weekly working time resources.

5.8 Appendix B: Detailed results

In Section 5.5, Tables 5.3 to 5.5 reported a summary of the results obtained by the proposed LNS algorithm for three different sets of rules. In this appendix, we provide in Table 5.7 the details of these results, namely, the number of vehicles used (NV) and the total distance traveled (TD) in the computed solution for each individual VRPTWDR instance and each set of rules. Again, we report best and average results over five runs.

	Basic rules				With working time rules				All rules			
	Best		Avg.		Best		Avg.		Best		Avg.	
	NV	TD	NV	TD	NV	TD	NV	TD	NV	TD	NV	TD
C101	10	931.37	10	931.37	10	931.37	10	931.37	10	931.37	10	931.37
C102	10	904.25	10	904.34	10	904.25	10	904.50	10	904.25	10	904.34
C103	10	833.92	10	833.92	10	833.92	10	833.92	10	833.19	10	833.19
C104	10	819.81	10	819.81	10	819.81	10	819.81	10	819.81	10	819.81
C105	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94
C106	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94
C107	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94
C108	10	827.38	10	827.38	10	827.38	10	827.38	10	827.38	10	827.38
C109	10	825.65	10	825.65	10	825.65	10	825.65	10	825.65	10	825.65
C201	6	882.23	6	916.36	6	881.50	6	891.41	5	810.05	5	834.29
C202	5	774.87	5	781.71	5	813.40	5	835.64	5	695.75	5	702.92
C203	4	665.10	4	665.10	5	698.13	5	703.58	4	661.84	4	662.28
C204	4	646.25	4	647.55	4	662.01	4	667.79	4	649.70	4	651.66
C205	4	639.22	4	639.27	5	684.42	5	685.56	5	678.70	5	681.02
C206	4	628.93	4	630.25	5	685.63	5	688.30	5	676.57	5	677.99
C207	4	646.50	4	647.16	5	693.97	5	698.44	5	674.67	5	675.40
C208	4	626.04	4	628.36	5	673.61	5	676.28	4	672.30	4	674.02
R101	9	1503.75	9.6	1400.33	10	1326.78	10	1326.93	9	1319.88	9	1324.66
R102	8	1227.28	8	1246.53	8	1263.11	8.4	1227.93	8	1177.31	8	1189.03
R103	8	972.26	8	972.29	8	1263.11	8	978.04	8	967.96	8	969.35
R104	8	859.68	8	864.03	8	868.88	8.2	870.21	8	855.72	8	860.05
R105	8	1113.99	8	1122.94	8	1116.68	8	1121.31	8	1090.69	8	1090.90
R106	8	1008.74	8	1016.19	8	1019.20	8	1022.87	8	998.35	8	1000.62
R107	8	900.93	8	902.01	8	902.76	8	904.89	8	892.90	8	896.49
R108	8	849.81	8	851.89	8	848.45	8	852.11	8	840.95	8	846.22
R109	8	928.61	8	930.14	8	928.61	8	931.69	8	923.28	8	923.60
R110	8	885.43	8	889.81	8	884.83	8	890.71	8	880.19	8	884.71
R111	8	881.69	8	890.21	8	882.07	8	888.77	8	881.27	8	886.33
R112	8	834.01	8	840.55	8	836.72	8	839.82	8	831.13	8	838.96
R201	7	1242.18	7	1248.90	7	1254.84	7	1266.72	7	1220.86	7	1232.59
R202	6	1108.22	6	1113.82	6	1116.22	6	1118.67	6	1093.46	6	1104.26
R203	5	968.26	5	976.45	6	918.82	6	922.62	5	957.70	5.8	916.71
R204	4	789.93	4	812.29	5	776.57	5	783.14	5	770.21	5	772.34
R205	5	1060.08	5	1080.76	6	1011.29	6	1021.03	6	1000.40	6	1003.45
R206	5	936.74	5	939.02	6	929.44	6	932.90	5	958.17	5.8	924.81
R207	4	863.80	4.4	858.25	5	857.74	5	865.09	5	840.61	5	850.21
R208	4	749.46	4	751.03	5	746.39	5	751.66	5	754.21	5	756.61
R209	5	940.29	5	949.57	6	905.62	6	911.78	5	950.53	5.8	911.95
R210	5	982.53	5	1002.79	6	943.64	6	948.29	6	938.69	6	942.13
R211	5	792.13	5	801.13	5	801.93	5	819.93	5	788.35	5	793.39
RC101	9	1303.24	9	1307.29	9	1305.09	9	1308.14	9	1293.82	9	1298.37
RC102	9	1180.67	9	1186.06	9	1186.64	9	1193.30	9	1177.51	9	1181.76
RC103	9	1084.10	9	1092.00	9	1082.37	9	1086.51	9	1085.66	9	1087.71
RC104	9	993.19	9	994.73	9	993.19	9	995.13	9	993.13	9	993.55
RC105	9	1220.76	9	1222.64	9	1224.41	9	1227.81	9	1203.34	9	1207.21
RC106	9	1095.18	9	1107.52	9	1093.62	9	1099.70	9	1093.96	9	1103.05
RC107	9	1033.79	9	1040.18	9	1029.58	9	1039.70	9	1028.11	9	1036.29
RC108	9	989.17	9	992.19	9	988.54	9	995.55	9	982.59	9	982.94
RC201	7	1424.54	7	1326.88	8	1384.01	8	1385.84	7	1395.15	7	1398.04
RC202	6	1247.44	6	1186.69	7	1193.12	7	1193.91	7	1153.45	7	1156.78
RC203	5	1120.97	5.4	949.37	6	1037.39	6	1040.52	6	1016.92	6	1020.71
RC204	5	859.40	5	792.51	5	877.17	5	885.75	5	863.22	5	869.84
RC205	7	1294.55	7	1294.63	7	1310.16	7	1312.01	7	1270.78	7	1273.89
RC206	6	1138.35	6	1150.42	6	1179.85	6.4	1166.05	6	1129.39	6.2	1133.85
RC207	6	1073.72	6	1075.92	6	1087.68	6	1089.52	6	1046.83	6	1056.93
RC208	5	863.02	5	869.05	5	906.90	5.8	865.35	5	850.63	5	862.71
Total	396	53460	397.4	53611	413	53419	414.8	53558	405	52665	407.6	52771

Table 5.7 Detailed results obtained by the proposed LNS algorithm

CHAPITRE 6

METAHEURISTICS FOR AN OIL DELIVERY VEHICLE ROUTING PROBLEM

Article soumis à *European Journal of Operational Research* (Octobre 2010) et écrit par:

ERIC PRESCOTT-GAGNON

École Polytechnique de Montréal

GUY DESAULNIERS

École Polytechnique de Montréal

LOUIS-MARTIN ROUSSEAU

École Polytechnique de Montréal

Abstract

Companies distributing heating oil typically solve vehicle routing problems on a daily basis. Their problems may involve various features such as a heterogeneous vehicle fleet, multiple depots, intra-route replenishments, time windows, driver shifts and optional customers. In this paper, we consider such a rich vehicle routing problem that arises in practice and develop three metaheuristics to address it, namely, a tabu search (TS) algorithm, a large neighborhood search (LNS) heuristic based on this TS heuristic and another LNS heuristic based on a column generation (CG) heuristic. Computational results obtained on instances derived from a real dataset indicate that the LNS methods outperform the TS heuristic. Furthermore, the LNS method based on CG tends to produce better quality results than the TS-based LNS heuristic, especially when sufficient computational time is available.

Keywords: Oil delivery, rich vehicle routing, large neighborhood search, tabu search, column generation.

6.1 Introduction

This paper addresses a real-life application arising in the heating oil (or propane) distribution industry. In this application, an oil distributor supplies a set of customers with a single oil product that is stored at each customer in an oil tank of a known capacity. For most of these customers (called the VMI customers, for vendor-managed inventory), the oil inventory is managed by the distributor which must ensure, as much as possible, no oil shortages. The other customers are called *spot* customers because they can request a delivery at any time. When such a request is accepted by the distributor, it must fulfill it within a predetermined time limit (for instance, within 24 or 48 hours). Without spot customers, the problem corresponds to an inventory routing problem (IRP) in which the customers to service every working day must be determined along with the vehicle delivery routes and the oil quantity to deliver to each serviced customer. In practice, the size of such an IRP makes it intractable. Indeed, a real-life instance (for a region) can involve between 3 to 20 delivery vehicles and between 4,000 and 30,000 customers, each vehicle visiting between 30 and 50 customers per day, and each customer being serviced approximately at every 40 days during winter. Furthermore, the future customer demands are difficult to forecast because they highly depend on the forthcoming temperatures which are difficult to predict. Consequently, in the industry, this planning problem is tackled as a sequence of one-day problems. This allows to deal with spot customers and to avoid inaccurate future demand forecasts.

Nowadays, the distributors rely on sophisticated forecasting systems (based on the historical consumption of each customer and past daily temperatures) to provide, at a given day of

the year, a good estimate of the oil remaining in the tank of every of its VMI customers. Based on these estimates, a distributor can determine which of its VMI customers must receive a delivery on the next day. Typically, a customer qualifies for a delivery if the estimate of oil remaining in its tank is about to fall below a safety stock level (say below 20% of the volume of its tank on the next day). The VMI and spot customers that must receive a delivery on the next day are called the *mandatory* customers. Those that will become mandatory in the following few days are called the *optional* customers. If these customers are located nearby some of the mandatory customers, it might be profitable to visit them on the same day even if their estimated oil inventory is not considered low enough for triggering a delivery request. To evaluate the profitability of visiting an optional customer, one must thus approximate the future refill cost (detour) that would be incurred if it was visited on a subsequent day.

The one-day problem considered in this paper can be briefly stated as follows. Given a set of mandatory customers, a set of optional customers with their estimated future detours, find vehicle routes that visit all mandatory customers and possibly optional customers such that the total traveled distance minus the sum of the saved detours of the visited optional customers is minimized. This vehicle routing problem (VRP) includes several additional features such as customer time windows, multiple vehicle depots, a heterogeneous fleet of vehicles, driver shifts, and intra-route vehicle replenishments that will be detailed later. We call this rich VRP the *oil delivery vehicle routing problem* (ODVRP).

The classical VRP and several of its variants have been widely studied (Cordeau *et al.*, 2007; Golden *et al.*, 2008). As surveyed in Dror (2005), several papers have addressed oil and propane distribution. The closest works to ours have been realized by Dror *et al.* (1985) and Dror et Ball (1987) who modeled a propane delivery problem as an IRP defined over a short horizon (such that no customers are serviced twice in the horizon) and involving mandatory and optional customers. In this problem, the mandatory customers must be serviced once before a deadline in the horizon. The objective function includes traveling costs and future costs that favor servicing the mandatory customers as late as possible to avoid too frequent deliveries and the optional customers within the horizon to avoid a costly detour in a near future. These authors develop a two-stage solution method. In the first stage, the customers are assigned to delivery days (optional customers might remain unassigned) using linear programming and a rounding procedure. In the second stage, a VRP is solved for each day using a construction heuristic. This approach has not been designed to handle all the complexifying features (spot customers, time windows, driver shifts, intra-route replenishments, etc.) that need to be considered in the ODVRP.

In the following two decades, the research has focused on IRP with stochastic demands given that the future demands highly depend on the forthcoming temperatures. To deal

with these problems, several authors (Kleywegt *et al.*, 2002; Adelman, 2004, among others) developed Markov decision process models and corresponding approximate solution methods. Such solution approaches yield much higher computational times than the deterministic ones, which makes them difficult to use in practice for large instances involving complex features. The interested reader can consult the recent survey papers of Moin et Salhi (2007) and Bertazzi *et al.* (2008) for additional references on the IRP.

When considering the sequence of ODVRP to solve over a short horizon as a single problem, one can observe similarities between this problem and the multi-period VRP that consists of determining vehicle routes for each period such that each customer is serviced once, each within a predefined subset of the periods. Bostel *et al.* (2008) proposed a metaheuristic and a column generation method for this problem that does not consider optional customers, spot customers, and intermediate vehicle replenishments. Intermediate replenishments were taken into account in different VRP variants by Angelelli et Speranza (2002), Tarantilis *et al.* (2008) and Crevier *et al.* (2007) using different methodologies.

The goal of this paper is to address a problem that matches with the current industry practice and to propose solution methods that can solve real-life instances in relatively fast computational times. Its main contribution is thus to develop different heuristics for solving the ODVRP and compare their performances. We introduce three metaheuristics: a tabu search (TS) heuristic and two large neighborhood search (LNS) methods that rely, respectively, on the TS heuristic and on a CG heuristic for exploring the neighborhoods. The reasons for favoring these methods are as follows. TS is well known for producing in fast computational times good quality solutions to various complex VRP. In the last decade, LNS algorithms have also proven to be successful for several types of VRP. In particular, LNS combined with CG was used by Prescott-Gagnon *et al.* (2009, 2010) for the VRP with time windows (VRPTW) and the VRPTW with driver rules.

The paper is organized as follows. The ODVRP is formally stated in Section 6.2. How to compute the estimated detours of the optional customers is discussed in Section 6.3. The three proposed metaheuristics are described in Sections 6.4 to 6.6. Computational results comparing these three methods on instances derived from a real-life dataset are reported in Section 6.7. Finally, conclusions are drawn in Section 6.8.

6.2 Problem statement

Given a heterogeneous fleet of vehicles housed in a set of depots $\mathcal{K}$, a set of customers $\mathcal{N}$ divided into a subset of mandatory customers $\mathcal{N}_m$ and a subset of optional customers $\mathcal{N}_o$, the ODVRP consists of determining feasible routes to deliver one type of heating oil to a

subset of the customers in $\mathcal{N}$ on a specific day T. Every mandatory customer in $\mathcal{N}_m$ must be visited exactly once, whereas every optional customer in $\mathcal{N}_o$ must be visited at most once. When an optional customer $i \in \mathcal{N}_o$ is visited on day T, it incurs a bonus β_i corresponding to an approximation of the traveled distance saved for not visiting it in one of the subsequent days. The procedure that we use for computing these bonuses is exposed in the next section. The objective of the ODVRP is minimizing the total distance traveled by the vehicles (which is usually proportional to the total traveling cost) minus the bonuses of the visited optional customers. The following constraints and features must be taken into account.

When planning the routes for day T, the quantity of oil q_i to be delivered to a customer $i \in \mathcal{N}$ is given by the difference between the capacity of the customer's tank and the estimated oil inventory in that tank, that is, the distributor uses an order-up-to level policy. As it is the case in general, we assume that q_i is relatively small compared to the capacity of any vehicle and, therefore, a single delivery can always fulfill a customer demand. Certain customers (mostly commercial customers) are subject to delivery time restrictions that are imposed, for instance, by the inaccessibility of their oil tank outside their business opening hours. Such a restriction for a customer i is modeled as a time window $[a_i, b_i]$ that indicates the admissible start of service times at this customer. When no time restrictions apply to a customer, its time window is set as the whole day T. Note that a driver can arrive before time a_i at customer i and wait until a_i before starting service. Every customer $i \in \mathcal{N}$ also has an associated service time s_i that can depend on the quantity delivered.

Let $\mathcal{E}$ be the set of drivers available on day T. Each driver $e \in \mathcal{E}$ is assigned to a depot $k_e \in \mathcal{K}$ (which can be his home in a rural area), a working shift defined by a time interval $[a_e, b_e]$ and a pre-assigned vehicle v_e of given capacity Q_e that must be picked up at the driver's depot and returned there. At most one route can be assigned to each driver $e \in \mathcal{E}$ and this route must start and end at depot k_e, must not exceed capacity Q_e (bearing the replenishment possibilities described below) and its time span must be included in shift $[a_e, b_e]$. Note that, because the vehicles are pre-assigned to the drivers, there is no need to consider vehicle availability constraints.

Each day and often in a middle of a shift, the vehicles need to be replenished. Replenishments must be performed at a replenishment station, which can correspond to a depot or to an arbitrary location. Denote by $\mathcal{F}$ the set of replenishment stations and by s_f^e the time required at station $f \in \mathcal{F}$ to replenish vehicle v_e of driver $e \in \mathcal{E}$. In general, when the vehicles start from a depot harboring a replenishment station, the vehicles are fully replenished before the beginning of the driver shifts. In other cases, they might be partially loaded or even empty. For simplicity reasons, we assume in this paper that they are fully loaded.

Let $\mathcal{L} = \mathcal{K} \cup \mathcal{F} \cup \mathcal{N}$ be the set of all locations (depots, replenishment stations, and

customers). With every pair of locations $(i, j) \in \mathcal{L}^2$ that can be visited consecutively, a travel time t_{ij} and a travel distance d_{ij} are associated. The travel time t_{ij} is actually equal to the time required to go from location i to location j plus the service time, if any, at location i. The travel distance and the travel time are clearly related but the latter often depends on the types of road traveled (street, boulevard, highway, etc.) and the expected congestion, or simply on the expected average speed.

To summarize, a route for a driver e starts from depot k_e, visits a sequence of customers, possibly replenishing from time to time, before returning to the same depot. This route is feasible if it respects the driver's shift $[a_e, b_e]$ and the time window $[a_i, b_i]$ of each visited customer i and if the total amount of oil delivered between two consecutive replenishments (including those performed before the start and after the end of the route) does not exceed the vehicle capacity Q_e. The total distance traveled along this route is given by the sum of the travel distances d_{ij} between every pair (i, j) of consecutive locations visited along the route.

6.3 Optional customer bonuses

When the travel distances respect the triangle inequality, visiting an optional customer on day T can only increase the total traveled distance. However, it might be profitable to make a small detour to service an optional customer if it can be feasibly inserted into a route and if postponing its visit to a subsequent day yields a larger detour later on. The detour incurred by visiting a customer l between two other locations i and j is given by $d_{il} + d_{lj} - d_{ij}$ (i.e., the additional distance traveled to visit l between i and j). Consequently, when planning the routes of day T, one can take into account possible distance savings for the next few days by offsetting the detour incurred for servicing an optional customer l by its bonus β_l, which should estimate the detour incurred by a visit on a later day.

One intuitive way that can be used for computing the optional customer bonuses is to solve an ODVRP instance for each of the following few days, considering in each instance only the customers becoming mandatory on the corresponding day. The detours (bonuses) could then be computed directly from the solution obtained for each day. This approach has two main disadvantages. First, it is computationally expensive. Second, the computed detours do not take into account the fact that several optional customers can be serviced on day T. Indeed, if it turns out that the detour of a customer l is computed using (optional) customers i and j as adjacent customers, then this detour value is obsolete if customer i or j is serviced on day T. A different approach was proposed by Dror et Ball (1987). They suggest to define a neighborhood for each customer and to compute the detour yielded by a

customer in the shortest Hamiltonian tour visiting the customers in its neighborhood. Then, for each customer, a second detour is computed assuming that this customer is adjacent to the depot in a route. Both detours are then averaged to give a final detour value. Assuming relatively small neighborhoods, this approach is much faster than the previous one. However, it also computes detours without taking into account that several optional customers can be serviced on day T.

To alleviate this difficulty, we develop a simple procedure for computing the optional customer bonuses. Given that it is not possible to compute precisely the detour incurred by a customer l if it is not visited on day T, we propose to use a weighted average of the detours for all pairs of customers i and j that can yield the real detour. Weights are used for two reasons. First, it is difficult to determine which pairs should be considered and weights allow to consider them all, each with a relative importance. Second, the detour for a customer l yielded by customers i and j that are remotely located from l can be very small compared to the one produced by customers in the vicinity of customer l as illustrated in Figure 6.1. In this figure, customers i_1 and j_1 are closer to customer l than customers i_2 and j_2 are, but they yield a larger detour because i_2, j_2 and l are almost colinear. As a customer is, more often than not, serviced between customers that are in its vicinity, attributing small (resp. large) weights to detours generated by remotely (resp. closely) located customers provides a better estimate of the expected detour.

The proposed procedure is as follows. Let l be a customer in $\mathcal{N}_o$ and let $\mathcal{P}^l$ be the list of all pairs of customers i and j that can yield the real detour for l, that is, all pairs (i, j) such that $i, j \in \mathcal{N}_o$, $i \neq j$, $i \neq l$ and $j \neq l$. First, for each pair $(i, j) \in \mathcal{P}^l$, compute the sum of the distances $D^l_{ij} = d_{il} + d_{lj}$ between l and the two customers i and j. Second, sort the pairs $(i, j) \in \mathcal{P}^l$ in increasing order of D^l_{ij}. Third, compute the bonus β_l as follows:

$$\beta_l = w \sum_{p=1}^{|\mathcal{P}^l|} (1-w)^{p-1} (d_{i_p l} + d_{l j_p} - d_{i_p j_p}), \tag{6.1}$$

where $w \in (0, 1)$ is a parameter whose value provides the weight of the first detour, index p gives the order of the pair of customers in the ordered list $\mathcal{P}^l$, and (i_p, j_p) denotes the pair of customers of rank p. With this formula and $w = 10\%$, the weights of the detours yielded by the first four pairs are 10.00%, 9.00%, 8.10%, 7.29%. Note that the right-hand side of (6.1) is not (exactly) a weighted average of the detours because the series is not infinite. However, in practice, $|\mathcal{P}^l|$ is large enough to yield a sum of the weights ($w(1-w)^{p-1}$) very close to one. Note also that we have chosen to not consider the depots and the replenishment stations in this procedure because, in practice, they are not often visited in a route (approximately once

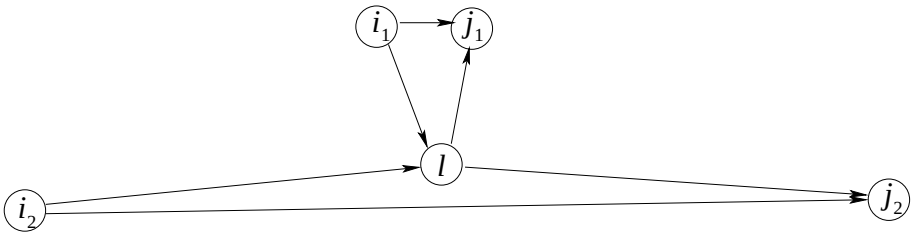

Figure 6.1 Two different detours for customer l

every 15 customers).

6.4 Tabu search heuristic

To solve the ODVRP, we first introduce a TS algorithm. TS (see Glover and Laguna, 1997) is a metaheuristic that has been successful at solving a wide variety of combinatorial optimization problems, including many VRP variants (Cordeau *et al.*, 2001b). It is an iterative method that starts from an initial solution and applies local modifications (moves) to improve it. Possible moves can be defined by a set of operators and are generally quite simple. A neighbor is a solution that differs from a current solution by only one move, and at each iteration, the move creating the best neighbor is chosen even if the objective value deteriorates. To avoid cycling, a memory of past moves, often called the tabu list, is kept in order to forbid recent moves to be reversed for a number of iterations. This allows the search to escape from local minima.

Algorithm 6.1 presents the pseudo-code of a generic tabu procedure when there are multiple move types. It requires an initial solution Sol_0 and a parameter indicating the total number of TS iterations $TotTsIter$ to perform. The process starts by initializing the tabu list and an iteration counter i. Every possible non tabu move of every move type is evaluated on the current solution Sol_i. This evaluation computes the cost difference between the cost of the solution that would result from this move and the cost of Sol_i. This cost difference is stored in Move.Cost. The move yielding the best cost is then applied to create a new solution Sol_{i+1} and added afterwards to the tabu list. Certain moves in the tabu list may also be removed from this list to limit the number of tabu moves. The process repeats with the new solution until reaching $TotTsIter$ iterations.

The length of the tabu list is defined by a parameter L_{tabu} but has a random aspect. When a move is applied to a given element of the problem, this element becomes fixed for a number of iterations selected in $L_{tabu} \pm 5\%$. For instance, if a move is to insert an unserved customer, no other moves will be applied to this customer for the selected number of iterations. An aspiration criterion is however applied (but not described in Algorithm 6.1): when a move

involving a tabu element generates a new overall best solution, it is accepted regardless of the tabu consideration.

The tabu search method that we propose for the ODVRP uses multiple move types to take into account the different aspects of the problem. They are:

- Remove a (mandatory or optional) customer from a route;
- Insert an unserved customer into a route;
- Exchange a customer from one route to another;
- Insert a visit to a replenishment station;
- Remove a visit to a replenishment station;
- Change the location of a replenishment;
- Exchange routes between two driver shifts.

The search is allowed to visit solutions that are infeasible because not all mandatory customers are visited, some time windows or driver shifts are violated, or vehicle capacity is exceeded. For a given solution x, its cost is given by $z(x) = d(x) - \beta(x) + \zeta q(x) + \alpha t(x) + \gamma m(x)$, where $d(x)$ is the total distance traveled, $\beta(x)$ is the sum of the bonuses of the visited optional customers, $q(x)$ and $t(x)$ are the total capacity and time window violations, and $m(x)$ is the total number of unserved mandatory customers. The computation of $q(x)$ and $t(x)$ is performed as suggested by Nagata *et al.* (2010). $q(x)$ is equal to the sum of the exceeding demands between two replenishments, while $t(x)$ is equal to the sum of the violations of the time window at each visited customer assuming (to avoid a cascading effect) that the time is reset to the time window lower bound when there is a violation (see Nagata *et al.*, 2010, for more details). The values of the parameters α, ζ and γ are adjusted dynamically throughout the solution process in order to obtain a *strategic oscillation* between feasible and infeasible solutions (see Glover et Laguna, 1997). They are all equal at the beginning of the algorithm and may change at every I^{adj} iterations. Each parameter value is adjusted separately considering the history of the solutions visited in the last I^{hst} iterations. The value of α is multiplied by 2 if the number of solutions in the history that are time-window infeasible exceeds ρ^{max} and divided by 2 if it is less than ρ^{min}, where ρ^{max} and ρ^{min} are predetermined parameters. The same adjustment procedure applies for parameters ζ and γ. Keeping a relatively small number of infeasible solutions with respect to each criterion ensures that the overall number of infeasible solutions is reasonable. To achieve this in our computational experiments, we used the following parameter values: $I^{adj} = 20$, $I^{hst} = 100$, $\rho^{max} = 25$, and $\rho^{min} = 15$.

The initial solution Sol_0 is found using a greedy heuristic. This heuristic computes sequentially a route for each driver in an arbitrary order. Such a route is build greedily by selecting the next customer to service as the one that can be serviced the earliest among the

Algorithm 6.1 Generic tabu search algorithm

Require: Initial solution Sol_0
Require: Total number of iterations $TotTsIter$
  BestSol $\leftarrow \text{Sol}_0$
  Create an empty tabu list
  $i \leftarrow 0$
  repeat
    BestCost $\leftarrow \infty$
    for all MoveType $\in$ MOVETYPES **do**
      for all Move $\in$ MoveType.MOVES **do**
        if Move.Cost $<$ BestCost and Move is not tabu **then**
          BestMove $\leftarrow$ Move
          BestMoveType $\leftarrow$ MoveType
          BestCost $\leftarrow$ Move.Cost
    $\text{Sol}_{i+1} \leftarrow$ BestMoveType.ApplyMove(BestMove, Sol_i)
    Sol_{i+1}.Cost = Sol_i.Cost + BestCost
    Update tabu list (add BestMove and remove, if any, past moves becoming non tabu)
    if Sol_{i+1}.Cost $\leq$ BestSol.Cost *and* Sol_{i+1} is feasible **then**
      BestSol $\leftarrow \text{Sol}_{i+1}$
    $i \leftarrow i + 1$
  until $i = TotTsIter$
  Return BestSol

unserved mandatory customers as long as it is feasible with respect to the driver's shift (a feasible return to the depot must be possible), the customer's time window, and the vehicle capacity. When no customers can be serviced due to vehicle capacity, then the route is extended towards the closest replenishment station before adding further customers (if time permits). This heuristic does not guarantee that all mandatory customers can be serviced. However, to favor a wide coverage of these customers, no optional customers are included in the initial routes.

6.5 Large neighborhood search heuristic based on tabu search

Starting from an initial solution, an LNS method (Shaw, 1998) is an iterative method that alternately remove elements from the current solution (destruction step) and reinsert them (reconstruction step) in order to find, hopefully, a better solution. A neighborhood at a given iteration is the set of all solutions that contain the undestroyed parts of the current solution. Because the size of the neighborhood increases exponentially with the number of elements removed, it has the potential to change a large portion of the solution. The algorithm stops when a certain number of iterations has been performed.

In Algorithm 6.2, we present the pseudo-code of a generic LNS algorithm that relies on different operators in the destruction and the construction phases. It requires an initial solution Sol_0 and a parameter indicating the total number of LNS iterations $TotLnsIter$ to perform. It starts by initializing the iteration counter i. Then, it repeats the same steps: choose a destruction operator, destroy a part of the current solution Sol_i to yield a partial solution $\overline{\text{Sol}}_{i+1}$, choose a reconstruction operator, and reconstruct a complete solution Sol_{i+1}. If this solution is better than the best solution found so far, then it becomes the best solution found. The process repeats with solution Sol_{i+1} (even if it was not better than the previous solution Sol_i) until reaching $TotLnsIter$ iterations.

In this section, we propose a first LNS heuristic for the ODVRP that uses TS in the construction phase. This heuristic, denoted LNS-TS, starts from the initial solution computed by the greedy algorithm described in Section 6.4. The destruction procedure selects a fixed number of customers to remove from their current route using a destruction operator. The other customers are said to be fixed for this LNS iteration, that is, they must remain in their route except if such a customer is serviced between two selected (thus freed) customers in the current solution. In this case, this customer is also freed. The sequence of visits to fixed customers must remain the same. Furthermore, no customers can be inserted between two fixed customers that are adjacent in a current route. Note that visits to replenishment stations or depots are never fixed. Also, the destruction operator may select customers that were not served in the current solution. Unserved customers that are not selected are fixed in their unserved state and, therefore, cannot be inserted in any route in this iteration.

At each LNS iteration, one of four destruction operators is chosen by a roulette-wheel procedure (Pisinger et Ropke, 2007) which favors the selection of the most efficient operators according to the improvements they yielded in the past iterations (for complete details, see Prescott-Gagnon *et al.*, 2009). The four operators are inspired from those of Prescott-Gagnon *et al.* (2009) and adapted to the ODVRP. They are as follows.

Time operator: For each customer, the time interval in which the customer can be visited without yielding an infeasible route is first computed. For an unserved customer, this interval is simply its time window or the whole planning horizon (day T) if there is no time window. Then, a time τ is selected randomly within the planning horizon. Finally, customers are selected randomly favoring those with time intervals containing τ or close to τ.

Proximity operator (Shaw, 1998): A seed customer is first selected randomly. The other customers are selected randomly, favoring those located near the location of the seed customer.

Longest detour operator: Customers are selected randomly, favoring those who generate a longer detour in the current solution. The detour associated with an unserved customer

Algorithm 6.2 Generic large neighborhood search algorithm

Require: Initial solution Sol_0
Require: Total number of iterations $TotLnsIter$
  $i \leftarrow 0$
  BestSol $\leftarrow \text{Sol}_0$
  repeat
    OpDestroy $\leftarrow$ ChooseDestructionOperator()
    $\overline{\text{Sol}}_{i+1} \leftarrow$ Destroy(Sol_i,OpDestroy)
    OpReconstruct $\leftarrow$ ChooseReconstructionOperator()
    $\text{Sol}_{i+1} \leftarrow$ Reconstruct($\overline{\text{Sol}}_{i+1}$,OpReconstruct)
    if Sol_{i+1}.cost $<$ BestSol.cost **then**
      BestSol $\leftarrow \text{Sol}_{i+1}$
    $i \leftarrow i + 1$
  until $i = TotLnsIter$
  Return BestSol

i corresponds to its bonus β_i.

Smart operator (Rousseau *et al.*, 2002): A first seed customer is selected. If it is serviced in the current solution, some of its adjacent customers in its route are also selected. A next seed customer is selected randomly, favoring those located near the location of the first seed customer. Once again, if this customer belongs to a route, some customers adjacent to it are also selected. The process is repeated until selecting the right number of customers. Note that once a customer and its adjacent customers are selected, no more customers from the same route can be selected.

The TS method of Section 6.4 is used in the reconstruction phase. At a given LNS iteration, the TS method starts from the incumbent solution and applies the moves presented in Section 6.4 but only to the parts of the solution that were freed in the destruction phase. The TS method is thus limited to the neighborhood defined for this LNS iteration.

After a fixed number of TS iterations, the TS heuristic returns the best feasible solution found to the LNS algorithm which then becomes the new incumbent solution. It may happen that the solution returned is the initial solution. This does not imply that the LNS algorithm will stall forever as the destruction phase will destroy different parts of the solution in the next iterations. Nevertheless, stalling was observed in preliminary tests. This led us to introduce a diversification strategy that is invoked when the TS method is unable to improve the current solution after a fixed number n_{same} of LNS iterations: instead of returning the best feasible solution found, the TS heuristic returns the last feasible solution visited. This strategy was devised in the hope of moving the search to a different feasible region of the search space instead of going back to the same region over again. Through the tuning of parameter

n_{same}, we can achieve a suitable trade-off between intensification and diversification. For our computational experiments, n_{same} was set to 5.

Note that the LNS framework can be viewed as a guiding procedure for the TS method, restricting the number of TS moves at each iteration and leading it to promising areas of the search space.

6.6 Large neighborhood search heuristic based on column generation

The second LNS heuristic that we propose is denoted LNS-CG and differs from the LNS-TS heuristic only by the reconstruction phase, where reconstruction is performed by a CG heuristic instead of a TS heuristic. CG (see Desrosiers et Lübbecke, 2005) is one of the leading methodology for solving exactly various constrained VRP and crew scheduling problems. As for all exact methods, it can easily be transformed into a heuristic. To tackle the ODVRP restricted to a neighborhood, we propose to adapt the column generation heuristic of Prescott-Gagnon *et al.* (2009) that was developed for the VRPTW and embedded into an LNS framework to produce very competitive results on well-known VRPTW benchmark instances. For reasons of clarity, we first describe a column generation heuristic for the whole ODVRP. Afterwards, we discuss how this heuristic can take into account the neighborhoods imposed by the LNS framework.

The ODVRP can be modeled as a set partitioning type problem where all the variables are associated with feasible routes. Let $\mathcal{R}^e$ be the set of all feasible routes for driver $e \in \mathcal{E}$. For each route $r \in \mathcal{R}^e$, denote by d_r the total distance traveled along r, β_r the total of the bonuses collected along r, and n_{ri} the number of times (0 or 1) that customer $i \in \mathcal{N}$ is serviced along r. Furthermore, define a binary variable θ_r^e that is equal to 1 if route r is assigned to driver e and 0 otherwise.

Using this notation, the ODVRP can be formulated as follows.

$$\text{Minimize} \qquad \sum_{e\in\mathcal{E}} \sum_{r\in\mathcal{R}^e} (d_r - \beta_r)\theta_r^e \tag{6.2}$$

$$\text{subject to:} \qquad \sum_{e\in\mathcal{E}} \sum_{r\in\mathcal{R}^e} n_{ri}\theta_r^e = 1, \qquad \forall\, i \in \mathcal{N}_m \tag{6.3}$$

$$\sum_{e\in\mathcal{E}} \sum_{r\in\mathcal{R}^e} n_{ri}\theta_r^e \leq 1, \qquad \forall\, i \in \mathcal{N}_o \tag{6.4}$$

$$\sum_{r\in\mathcal{R}^e} \theta_r^e \leq 1, \qquad \forall\, e \in \mathcal{E} \tag{6.5}$$

$$\theta_r^e \text{ binary}, \qquad \forall\, e \in \mathcal{E},\ r \in \mathcal{R}^e. \tag{6.6}$$

The objective function (6.2) aims at minimizing the total traveled distance minus the bonuses of the optional customers serviced. Set partitioning constraints (6.3) ensure that each mandatory customer is visited exactly once by a driver, whereas set packing constraints (6.4) impose a maximum of one visit to each optional customer. Constraints (6.5) specify that at most one route can be assigned to each driver. Finally, (6.6) enforce binary requirements on the variables.

In practice, model (6.2)–(6.6) contains a huge number of variables. To overcome this difficulty, column generation is used to solve its linear relaxation, called the master problem in this context. This iterative method solves at each iteration a restricted master problem (RMP) and subproblems. The RMP is simply the master problem restricted to a subset of its variables. Solving it provides a primal and a dual solution. This dual solution is transferred to the subproblems that aim at finding negative reduced cost variables (columns). When such columns are found, they are added to the RMP before starting a new iteration. This iterative process is repeated until no negative reduced cost variables can be generated. In an exact algorithm, one must prove that there are no more negative reduced cost variables before stopping with an optimal master problem solution (the current RMP optimal solution). In our context, the subproblems are solved heuristically which means that the last RMP solution might not be optimal for the master problem.

The subproblems correspond to elementary shortest path problems with resource constraints (ESPPRC), where resources (see Irnich et Desaulniers, 2005) are used to impose the capacity and time window constraints, and elementarity is not required for replenishment stations that can be visited multiple times along the same route. There is one subproblem per driver $e \in \mathcal{E}$. Its underlying network $\mathcal{G}^e = (\mathcal{V}^e, \mathcal{A}^e)$ can be defined as follows. The node set contains $|\mathcal{N}|+|\mathcal{F}|+2$ nodes: one node for each customer $i \in \mathcal{N}$; one node for each station $f \in \mathcal{F}$; and a pair of source node o_1^e and sink node o_2^e representing the driver's depot at the beginning and the end of the driver's shift, respectively. The arc set $\mathcal{A}^e$ contains: start arcs (o_1^e, j), $\forall j \in \mathcal{N}$; end arcs (i, o_2^e), $\forall i \in \mathcal{N}$; travel arcs (i, j), $\forall i, j \in \mathcal{N}$ such that customer j can be visited immediately after customer i in at least one feasible route (that is, if $a_i + t_{ij} \leq b_j$ and $q_i + q_j \leq Q_e$); to replenishment arcs (i, f), $\forall\, i \in \mathcal{N}$ and $f \in \mathcal{F}$; and from replenishment arcs (f, i), $\forall\, i \in \mathcal{N}$ and $f \in \mathcal{F}$. With each arc $(i, j) \in \mathcal{A}^e$ is associated its corresponding travel distance d_{ij}.

At a given CG iteration, the subproblem for driver e, $e \in \mathcal{E}$, searches for the least reduced cost variable θ_r^e, $r \in \mathcal{R}^e$. The reduced cost $\bar{c}_r^e$ of such a variable is given by

$$\bar{c}_r^e = d_r - \beta_r - \sigma_e - \sum_{i \in \mathcal{N}} n_{ri}\pi_i = \sum_{(i,j) \in r} d_{ij} - \sum_{(i,j) \in r \mid i \in \mathcal{N}_o} \beta_i - \sigma_e - \sum_{(i,j) \in r \mid i \in \mathcal{N}} \pi_i,$$

where π_i, $i \in \mathcal{N}$, are the dual variables of the constraints (6.3) and (6.4), σ_e is the dual variable of the constraint (6.5) for driver e, and $(i,j) \in r$ means an arc $(i,j) \in \mathcal{A}^e$ belonging to the path in $\mathcal{G}^e$ representing route r. Consequently, in the ESPPRC subproblem for driver e, the (reduced) cost $\bar{c}^e_{ij}$ of the arc $(i,j) \in \mathcal{A}^e$ is equal to

$$\bar{c}^e_{ij} = \begin{cases} d_{ij} - \sigma_e & \text{if } i = o^e_1 \\ d_{ij} - \pi_i & \text{if } i \in \mathcal{N}_m \\ d_{ij} - \beta_i - \pi_i & \text{if } i \in \mathcal{N}_o \\ d_{ij} & \text{otherwise} \end{cases}$$

to ensure that the sum of the reduced costs of the arcs of a path is equal to the reduce cost of the corresponding route.

The ESPPRC is usually solved by dynamic programming (see Irnich et Desaulniers, 2005) which can be highly time-consuming when the time windows are wide as it is the case in our experiments. Consequently, we propose to solve the ESPPRC subproblems using a tabu search algorithm similar to the one introduced by Desaulniers *et al.* (2008) for generating rapidly columns in an exact CG method for the VRPTW. At every CG iteration, all columns in the current optimal basis for the RMP (they all have a zero reduced cost) are considered as initial solutions for the tabu search algorithm which performs a fixed number of tabu search iterations for each initial solution. The tabu moves considered here are a subset of the ones defined for the tabu search method described in the previous section because individual routes are sought when solving a subproblem, not a whole set of routes as for the ODVRP. They are: insert or remove a customer from a route, insert or remove a replenishment station from a route, and change the driver assigned to a route. Note that this last operator allows to switch from one subproblem to another for the same initial solution. In this tabu search algorithm, a move can be accepted only if it yields a feasible solution (route).

To derive an integer solution, the CG algorithm is embedded into a rounding procedure. After solving a linear relaxation whose computed solution is fractional, the variable θ^e_r with the largest fractional value is fixed at 1, defining a new linear relaxation that is also solved by CG.

In the LNS-CG algorithm, this CG heuristic is limited to the neighborhood defined for the current LNS iteration. Sequences of fixed customers are treated as aggregated customers in the subproblems, yielding routes that always respect the fixed parts of the current solution. Note that artificial variables are added to the covering constraints (6.3) of the mandatory customers in the RMP. This allows to not cover all these customers bearing a high penalty. This is essential when starting the solution process with an initial solution Sol_0 that does not visit all mandatory customers.

As in Prescott-Gagnon *et al.* (2009), we keep in memory a pool of columns that contains all the routes generated throughout the whole LNS solution process. At the start of a new LNS iteration, all columns in this pool that respect the fixed parts of the current solution are directly added to the RMP. This pool of columns thus acts as a long term memory mechanism.

6.7 Computational experiments

To test the proposed heuristics, ODVRP instances were derived from a database of an oil distribution company containing more than 4,000 customers. For each customer, the database provides its location, its tank size, its safety stock level, and a history of its last deliveries (delivered quantities and dates for up to 10 deliveries). To service these customers, the company relies on four drivers that are preassigned to vehicles of different capacities. These drivers operate out of two depots equipped with replenishment facilities. There are also three additional replenishment stations scattered through the serviced region, which comprises a rural part. The drivers have all the same shift. Distances and travel times between every pair of locations were computed using a geographical information system (GIS).

Using the previous data, two types of instances were devised. The first type (Section 6.7.1) considers a single day of operation, while the second (Section 6.7.2) considers a whole week of planning for which a sequence of ODVRP will be solved as in a rolling horizon procedure.

6.7.1 Single day

To create the daily test instances, we used the following methodology. First, three dates covered by the database and relatively far apart were selected. For each of these dates, an estimation of the volume of oil q_i consumed by each customer i since its last refueling was computed using linear interpolation through the history of deliveries. Customers were then sorted in decreasing order of the percentage of oil consumed with respect to their effective tank size (denoted U_i for customer i), which is equal to the tank size minus the safety stock level.

For each date, six instances involving the first 250 customers as the set of customers $\mathcal{N}$ were created. They differ by their subset of mandatory customers or their service and replenishment times. Three subsets of mandatory customers $\mathcal{N}_m$ are considered, containing the first 70, 85 and 100 customers. In each case, the remaining customers are optional customers. Two realistic settings are proposed for the delivery and replenishment times. In one setting, all service times s_i are fixed at 7 minutes and all replenishment times s_f^e at 10 minutes. In the other, they are all equal to 8 and 15 minutes, respectively. Table 6.1 summarizes the characteristics of the six instances for each date. Time windows (as

encountered in practice) were also imposed on 10% of the customers. Finally, following preliminary tests, the value of the parameter w used to compute the optional customer bonuses in formula (6.1) was set to 10%.

Larger instances involving the first 500 and the first 750 customers were also created for each date. Again, for each date and each instance size, six instances were devised. Compared to the 250-customer instances, the numbers of mandatory and optional customers are multiplied by 2 and 3 for the 500- and 750-customer instances, respectively. The two settings described above for the service and replenishment times are also considered (see Table 6.1). In order to simulate a company with a larger fleet, the available drivers and vehicles were duplicated once and twice, yielding 8 and 12 drivers, respectively. Because a larger company, typically, services more customers everyday that have a demand q_i very close to their effective tank size U_i, the demands derived from the database were also adjusted as follows: for the 500-customer (resp. 750-customer) instances, every computed demand q_i was increased by $\frac{U_i - q_i}{3}$ (resp. $\frac{2(U_i - q_i)}{3}$).

Overall, there are 18 instances (3 dates, 3 subsets of mandatory customers, 2 settings for service and replenishment times) for each instance size (250, 500, and 750 customers). They were all solved using each of the three proposed heuristics, namely, TS, LNS-TS, and LNS-CG. For the TS heuristic, the total number of iterations ($TotTsIter$) was set to 800,000 and the tabu list length L_{tabu} was fixed at 60% of the number of customers in the instance. For the LNS-TS heuristic, a total of $TotLnsIter = 1600$ LNS iterations were performed in which 1000 TS iterations were executed, yielding a total of 1,600,000 TS iterations. The neighborhoods were defined by removing 110, 180 and 240 customers for the 250-, 500- and 750-customer instances, respectively. Parameter L_{tabu} was fixed at 60% of the number of customers removed. Finally, for the LNS-CG heuristic, $TotLnsIter$ was also set to 1600 but the number of customers removed in the destruction phase was fixed at 70 for all instance sizes. Based on preliminary test results, we chose these parameter values so that each heuristic performed at its best given a targeted time limit (one hour for the 750-customer instances) that seems reasonable in practice. Because there is randomness in each proposed method, ten runs were executed for each instance with each method. All our experiments were conducted on an AMD Opteron processor clocked at 2.3GHz. The column generation heuristic in the LNS-CG heuristic was implemented using the Gencol library, version 4.5, and Cplex, verison 12.1, for solving the RMP.

The results of these experiments are reported in Table 6.2, which is divided horizontally into three parts, one for each instance size. Each part contains three rows, one for each method. The results in a row correspond to averages over the 18 instances solved ten times. In order, the columns indicate: the heuristic used, the best solution cost (i.e., total

	250 customers		500 customers		750 customers		Service	Replen.
ID	Mand.	Opt.	Mand.	Opt.	Mand.	Opt.	time (min)	time (min)
1	70	180	140	360	210	540	7	10
2	85	165	170	330	255	495	7	10
3	100	150	200	300	300	450	7	10
4	70	180	140	360	210	540	8	15
5	85	165	170	330	255	495	8	15
6	100	150	200	300	300	450	8	15

Table 6.1 Characteristics of the instances created for each date

distance minus awarded bonuses) obtained over the ten runs, the average solution cost over these runs, the total computational time, the number of optional customers visited, and the average number of replenishments per route. From these results, we make the following observations. The TS heuristic is clearly outperformed by the other two heuristics: it generates worst-quality solutions in higher computational times. Embedding it into an LNS framework provides much better results. In particular, it allows intensifying the search in medium-size neighborhoods and executing twice the number of TS iterations in less computational times. On the other hand, the LNS-CG heuristic yields the best solutions over all instances, and, in less computational times for the 500- and 750-customer instances. Consequently, with its global view of a neighborhood, the CG heuristic seems to be a better tool for reconstruction. As particularly obvious for the largest instances, better costs (obtained from the LNS-CG solutions) can be achieved when visiting less optional customers. This suggests that the TS and LNS-TS algorithms insert too many optional customers into the routes which do not give them much leeway afterwards to change the current solution and retrieve feasibility easily. As expected, the average number of replenishments per route is proportional to the total number of (mandatory and optional) customers visited. This total number indicates that a driver visits on average over 30 customers per day, which corresponds to practice (especially when the serviced region includes a rural part as is the case here).

For the 250-customer instances, Table 6.3 presents a breakdown per instance ID of the results obtained by the LNS-CG heuristic. For each instance ID (see Table 6.1), the same statistics as above are given. They correspond to averages over three instances (one for each date considered) solved ten times. From these results, one can observe, on the one hand, that longer service and replenishment times (in instance IDs 4 to 6) yield, as expected, costlier solutions because the problems are more constrained and less optional customers can be serviced by the available drivers (reducing the average number of replenishments per route). On the other hand, augmenting the number of mandatory customers (that is, from

Heuristic	Best cost	Avg cost	Time (sec)	Nb opt. cust. visited	Avg no. replen. per route
250 customers					
TS	264.3	279.1	1070	57.5	1.73
LNS-TS	251.2	265.4	717	56.3	1.67
LNS-CG	243.8	256.7	792	53.5	1.62
500 customers					
TS	391.0	414.8	3739	136.5	1.76
LNS-TS	377.9	403.4	2043	131.4	1.62
LNS-CG	374.3	400.5	1532	92.3	1.38
750 customers					
TS	544.3	586.1	8296	215.2	1.73
LNS-TS	534.8	581.9	3341	200.1	1.56
LNS-CG	522.8	549.3	2888	115.8	1.28

Table 6.2 Results for the single-day instances

instance 1 to instance 3, and from instance 4 to instance 6) also increases the solution cost for the same reasons and the average computational time because set partitioning constraints (6.3) for the mandatory customers are harder to satisfy than the set packing constraints (6.4) for the optional customers in a CG heuristic. Similar remarks can be made for the 500- and 750-customer instances.

In another series of experiments with the LNS-CG heuristic, we performed a sensitivity analysis on the number of customers removed in the destruction phase of each LNS iteration. We conducted these experiments only on the 250-customer instances. Again, $TotLnsIter$ was set to 1600 and ten runs were executed for each instance. The results are reported in Table 6.4. They show that increasing the size of the neighborhoods helps computing better-quality solutions and improving the robustness of the solution method. Indeed, the gap between the average cost and the best cost reduces as this size increases. On the other hand, the average computational time increases quite rapidly with the size of the neighborhoods.

6.7.2 Week planning

Presented in Section 6.3, the bonus β_i for an optional customer $i \in \mathcal{N}_o$ estimates the detour incurred by visiting this customer later than on day T. Consequently, if it is visited on day T, we consider that this detour is saved and can be subtracted from the objective function. These bonuses are thus incentives for servicing the optional customers on day T and

ID	Best cost	Avg cost	Time (sec.)	Nb opt. cust. visited	Avg no. replen. per route
1	204.2	209.9	703	73.1	1.66
2	241.2	255.2	798	60.6	1.68
3	269.6	287.1	896	46.7	1.73
4	209.2	219.3	689	61.1	1.54
5	252.2	265.7	778	47.3	1.57
6	286.1	303.0	885	32.0	1.56

Table 6.3 Breakdown of the LNS-CG results for the 250-customer instances

No. cust. removed	Best cost	Avg cost	Time (sec)	Nb opt. cust. visited	Avg no. replen. per route
50	248.2	262.9	681	52.7	1.65
70	243.8	256.7	792	53.5	1.62
90	242.0	252.5	1083	54.5	1.64
110	241.8	250.5	1409	54.0	1.64
130	240.1	246.9	1958	54.0	1.63

Table 6.4 LNS-CG results for different numbers of customers removed (250-customer instances)

minimizing the total distance traveled over a medium-term horizon if they estimate future detours adequately. To assess their impact on the medium-term solution quality, we propose to consider a one-week horizon and solve a sequence of ODVRP, one for each day of the week, over a rolling horizon spanning the week.

For these experiments, we created six one-week instances involving only VMI customers. For each of the three dates considered in the previous experiments, we selected the first 750 customers after sorting them as above. On day 1, the customer demands q_i are those computed by linear interpolation through the history of the deliveries. For the subsequent days, a fixed consumption rate, in percentage of the customer's tank capacity, is used for all the customers to determine the demands of the customers that are not visited earlier. For each date, two instances were created: one with 7-minute service times and 10-minute replenishment times, another with 8-minute service times and 15-minute replenishment times.

Each instance was solved using the following rolling horizon procedure. First, an ODVRP is built for the first day. The mandatory customers are those with a demand q_i above 95% of their effective tank size U_i and the optional customers are those that will meet this condition

in the next two days (assuming the fixed consumption rate stated above). This ODVRP is solved using the proposed LNS-CG heuristic. All customers serviced in the computed solution are then removed from the set of customers and a new ODVRP is defined for the next day. For this ODVRP, the sets of mandatory customers and optional customers are defined as for the first day (from the set of customers not serviced yet), using the adjusted demands of the customers. This ODVRP is solved and serviced customers are removed from the set of customers. This process repeats until reaching the last day of the week. Note that the optional customer bonuses must be recomputed each day according to formula (6.1). For these experiments, the LNS-CG heuristic is the only method used because it outperformed the other methods in Section 6.7.1. Its parameter setting is the same as the one applied for the single-day 250-customer instances.

Two series of experiments were conducted. In the first series, we evaluate the impact of the value of parameter w used in formula (6.1). Recall that this parameter controls the weight attributed to each pair of customers that can be adjacent to a customer i for which the bonus β_i is computed. A w value very close to 1 indicates that only the pairs that are the closest to customer i have a significant weight when computing this bonus. On the other hand, a value close to 0 means that a larger number of pairs has an impact in this computation. In the second series of tests, we assess the magnitude of the bonuses with respect to the traveled distances. If the bonuses are too high, then a large number of optional customers should be serviced even if this is not profitable overall. At the opposite, if they are too low, then larger detours will be incurred for visiting certain optional customers in the subsequent days.

For the first series of tests, the six one-week instances were solved using different values of w ranging from 0.01 to 1.0. Again, ten runs were performed for each instance. The quality of a solution is evaluated by the total distance traveled over the week. To ensure a fair comparison between the different parameter values, we force the covering of the same customers, namely, those that become mandatory during the week. Out of the 750 customers considered, there are 567 such customers on average. The other customers are, however, used to compute the bonuses of the optional customers. The results obtained are reported in Table 6.5 which provides for each value of w the following averages (taken over the best solution founds): the total distance traveled during the week to service the 567 customers, the total number of customers serviced on the day they were mandatory, and the total number of customers serviced as optional customers. These results indicate that a w value between 0.05 and 0.1 yields the least total distance. With a value of $w = 0.1$ (resp. $w = 0.05$), the detours for the first 22 (resp. 32) pairs of adjacent customers have a weight greater than 0.01 in the weighted average for computing the bonuses. For higher w values, the bonuses tend to underestimate the detours that can be saved in the future, resulting in less optional

customers serviced in advance. At the opposite, very low w values overestimate the optional customer bonuses, yielding more optional customers serviced in advance even if it does not worth it for many of them.

For the second series of tests, we first introduced a bonus multiplier μ that is applied to every bonus, that is, every bonus β_i, $i \in \mathcal{N}_o$, is replaced by $\mu\beta_i$. Then, we solved the six one-week instances (ten times each) with $w = 0.1$ and different μ values varying between 0 and 2. Again, only the customers becoming mandatory during the week must be serviced. Table 6.6 reports the results of these experiments. They show that formula (6.1) with $\mu = 1$ produces the best solutions, that is, those with the least total traveled distance. Hence, with $\mu = 1$, the bonuses seem to estimate with a sufficient accuracy the detours that will be incurred by visiting the optional customers later.

6.8 Conclusion

In this paper, we addressed the ODVRP that arises in the heating oil distribution industry. To solve this problem, we developed three metaheuristics: a TS algorithm and two LNS algorithms, one based on TS and the other based on CG. Computational results on instances derived from a real-life database showed that the LNS-CG heuristic outperforms the other two algorithms. On the other hand, the LNS-TS heuristic can be considered as a good alternative for a company that does not want to invest into a commercial linear programming solver that is required for the LNS-CG method. In another series of experiments that consisted of solving a sequence of ODVRP over a one-week rolling horizon, we showed that the bonuses used as incentives for covering the optional customers are adequate to minimize the total distance traveled over the week. As a future research direction, one can consider extending the proposed heuristics to treat a multiple product version of the ODVRP where the vehicles

	Total	No. cust. served as	
w	dist.	mand.	opt.
0.01	1982.5	196.0	371.0
0.025	1898.2	200.0	367.0
0.05	1847.5	227.5	339.5
0.10	1846.8	239.5	327.5
0.15	1856.3	255.0	312.0
0.25	1904.4	273.0	294.0
0.50	1943.1	288.1	278.9
1.00	2084.1	315.0	252.0

Table 6.5 LNS-CG results for different values of w (one-week instances)

μ	Total dist.	No. cust. served as mandatory	optional
0.00	2689.6	567.0	0.0
0.25	2176.4	429.7	137.3
0.50	1996.7	332.8	234.2
0.75	1900.5	282.5	284.5
1.00	1846.3	239.5	327.5
1.25	1907.0	233.3	333.7
1.50	2034.9	205.7	361.3
1.75	2068.6	171.5	395.5
2.00	2123.7	166.5	400.5

Table 6.6 LNS-CG results for different values of μ (one-week instances)

have several compartments of fixed sizes.

Acknowledgements: The authors wish to thank the personnel of Info-Sys Solutions (in particular, Yannick Grovalet, Asghar Mahmood and Michel Lafontaine) for proposing this problem to us and providing the data used for our computational experiments.

CHAPITRE 7

DISCUSSION GÉNÉRALE ET CONCLUSION

Dans cette thèse, une méthode hybride de recherche à grands voisinages utilisant la génération de colonnes heuristique pour explorer les voisinages a été proposée. Il est démontré qu'il est possible d'utiliser la génération de colonnes à l'intérieur d'un cadre métaheuristique et que cela peut être très compétitif sur des problèmes de tournées de véhicules par rapport aux méthodes de pointe de la littérature. La recherche à grands voisinages s'intègre de façon assez naturelle avec la génération de colonnes et permet d'accélérer grandement les temps de calcul en restreignant la génération de colonnes. C'est ce qui permet à cette méthode de se démarquer par rapport aux autres tentatives d'utiliser la génération de colonnes de façon heuristique. Bien que cette thèse se concentre sur les problèmes de tournées de véhicules, plusieurs autres types de problèmes, où la génération de colonnes est utilisable, pourraient bénéficier de la méthode présentée.

7.1 Contributions

L'objectif de cette thèse était de développer une méthode heuristique hybride efficace tirant profit de la puissance de la génération de colonnes pour résoudre un ensemble de problèmes de tournées de véhicules riches. La méthode devait pouvoir être appliquée sur des problèmes de grande taille.

D'abord, une méthode utilisant la génération de colonnes de façon heuristique à l'intérieur d'un cadre de recherche à grands voisinages a été proposée pour résoudre le VRPTW. Un ensemble de nouvelles stratégies avec d'autres déjà connues et la façon de les faire collaborer contribue à la méthode proposée qui obtient ainsi de résultats très compétitifs. Au moment des travaux, elle a réussi à obtenir de meilleurs résultats que toutes les autres méthodes publiées à ce moment. Elle a aussi réussi à améliorer la meilleure solution connue de 106 sur 356 instances connues de taille allant de 100 à 1000 clients.

Des stratégies pour considérer un ensemble de règles sur les horaires de chauffeurs à l'intérieur de la méthode ont ensuite été développées. Les règles considérées ont été mises en place par l'union européenne en avril 2007 (European Union, 2006) et doivent absolument être respectées. Très peu d'articles scientifiques travaux ont abordé ces règles et au début des travaux sur le sujet, aucune méthode ne considérait l'ensemble de ces règles sur des problèmes de très grande taille. La méthode utilisée sur le VRPTW a donc été généralisée. La

plus grande difficulté est de s'assurer que les tournées créées satisfont les règles. Pour vérifier la faisabilité des tournées, des stratégies à l'intérieur d'un algorithme d'insertion (méthode tabou pour résoudre le sous-problème) ont été élaborées. La méthode ainsi développée se démarque clairement des autres en permettant de diminuer le nombre de véhicules utilisés de 14% tout en réduisant la distance totale parcourue de 13,4% sur toutes les instances testées par rapport à la meilleure méthode de la littérature.

Finalement, la dernière contribution de cette thèse est de montrer que la méthode proposée peut être généralisée à un ensemble de contraintes provenant d'une application réelle. Le problème étudié, calqué sur les pratiques dans le domaine de la distribution d'huile de chauffage, contient un éventail assez grand de contraintes et de caractéristiques. La méthode est généralisée pour considérer un problème maître plus complexe et plusieurs sous-problèmes. L'apport d'une méthode de recherche à grands voisinages comme mécanisme de guidage pour une autre méthode heuristique comme la recherche tabou est aussi démontré.

7.2 Avantages et inconvénients de la méthode proposée

La méthode hybride proposée possède beaucoup de qualités de la génération de colonnes sur laquelle elle est basée. Par contre, elle possède plusieurs de ses défauts.

Il n'est pas toujours possible de trouver une formulation de génération de colonnes où il est possible de résoudre les sous-problèmes de façon efficace. La méthode est donc limitée aux problèmes où une telle formulation est possible. Ce groupe de problèmes est heureusement assez important pour que la génération de colonnes soit une méthode commercialement viable et pour que la méthode présentée soit pertinente. Les problèmes de tournées de véhicules entrent dans cette catégorie. Il y a beaucoup d'autres applications dans le domaine des transports, que ce soit en transport urbain, aérien, férroviaire ou maritime. On peut penser aussi à des problèmes de découpe (cutting-stock problems), d'ordonnancement d'atelier (job-shop scheduling), de création d'horaires de travail (crew scheduling problems), etc.

De plus, pour appliquer une méthode de génération de colonnes exacte, il est primordial de pouvoir résoudre le ou les sous-problèmes de façon efficace car cette opération sera répétée de nombreuses fois. Généralement, plus les sous-problèmes sont contraints, plus l'espace des solutions est restreint, ce qui amène la génération de colonnes à converger plus rapidement. Quand les sous-problèmes sont moins contraints, l'espace des solutions est plus grand et il devient beaucoup plus difficile de les résoudre de façon exacte.

En résolvant les sous-problèmes de façon heuristique, il est plus facile de considérer un plus grand éventail de problèmes. D'abord, les problèmes n'admettant pas de formulation de génération de colonnes où le sous-problème ne peut être résolu exactement de façon efficace

peuvent tout de même être considérés. Le problème considérant les règles européennes sur les horaires des chauffeurs présenté au chapitre 5 illustre parfaitement ce point. Le sous-problème est si difficile à résoudre de façon exacte qu'une méthode de génération de colonnes exacte est pour le moment impensable.

De plus, les instances qui peuvent être très difficiles à résoudre parce que les sous-problèmes ne sont pas très contraints peuvent être résolus beaucoup plus efficacement. On peut penser à la dernière instance connue de VRPTW de 100 clients dont on ne connaît toujours pas à ce jour la solution optimale (voir Baldacci *et al.*, 2010), qui est résolue de façon heuristique par la méthode présentée en des temps raisonnables. Bien qu'on puisse quand même traiter ces instances, l'espace des solutions des sous-problèmes est tout de même très grand, donc la génération de colonnes converge moins rapidement et engendre des temps de calcul plus élevés.

La plus lente convergence de la génération de colonnes engendre des temps de calcul qui peuvent être importants, notamment sur des instances de grande taille. C'est un des désavantages principaux de la méthode. Par contre, il a été illustré avec une analyse de sensibilité à la section 4.5.4 et dans les résultats du chapitre 6 (section 6.7) qu'en donnant plus de temps à la méthode, il est possible d'obtenir de meilleures solutions. Ceci est un avantage considérable sur d'autres méthodes qui vont plafonner sans l'apport de nouvelles stratégies. Toute amélioration qui permet d'accélérer les calculs, comme des machines plus rapides, a le potentiel de faire plus de calculs pour un même temps et ainsi obtenir de meilleurs résultats.

Un autre désavantage de la méthode est la nécessité d'avoir accès à du code de génération de colonnes et un solveur commercial de programmes linéaires tels que Cplex, Gurobi ou Xpress-MP. Il peut être concevable à partir d'un solveur commercial d'implémenter une méthode de génération de colonnes mais il s'agit d'une tâche importante. Par contre, pour quiconque possède un algorithme de génération de colonnes, la méthode est relativement simple à implémenter.

7.3 Améliorations futures

Plusieurs aspects de la génération de colonnes sont gérés de façon heuristique dans la méthode hybride présentée dans cette thèse : la résolution de la relaxation linéaire est heuristique ; les branchements sont heuristiques ; la résolution des sous-problèmes est heuristique. Pour chacun de ces aspects, des stratégies sont présentées. Ces stratégies sont souvent relativement simples, voir simplistes.

La méthode de branchement pour obtenir des solutions entières est basée seulement sur

la variable ayant la valeur fractionnaire la plus élevée. Il existe bien d'autres stratégies de branchement et il est fort probable que certaines d'entre elles permettent, soit d'obtenir de meilleures solutions entières, soit d'obtenir des solutions entières plus rapidement (branchement plus agressif) ou même idéalement les deux.

La méthode tabou utilisée pour résoudre les sous-problèmes est relativement simple et ne prend pas en compte la plupart des stratégies qui ont été implémentées pour d'autres méthodes tabous. La résolution des sous-problèmes pourrait donc être grandement améliorée par des méthodes plus rapides ou plus efficaces. La structure de la génération de colonnes fait en sorte qu'on a un grand nombre de solutions aux sous-problèmes en mémoire (variables dans le problème maître). Ces solutions forment en quelque sorte une population qui pourrait être à la base d'algorithmes génétiques. Il pourrait aussi être intéressant d'utiliser des techniques de programmation par contraintes afin de résoudre les sous-problèmes qui peuvent être contraints par le voisinage défini par le LNS en plus de par la nature du problème.

Quatre opérateurs sont utilisés pour déterminer les voisinages à explorer dans le LNS. Ces opérateurs dépendent de la structure du problème et certains ont du être adaptés d'un problème à l'autre. Il peut être intéressant de penser à de nouveaux opérateurs plus efficaces ou plus universels. L'accent dans cette thèse a surtout été mis sur la phase de reconstruction dans la recherche à grands voisinages. Il peut être bénéfique de se pencher de façon plus approfondie sur la phase de destruction. Il serait aussi possible, dans cette optique, d'étudier des stratégies pour faire varier la taille des voisinages définis par le LNS tout au long du processus au lieu de retirer toujours le même nombre de clients.

Au chapitre 6, une méthode tabou est présentée pour reconstruire les voisinages du LNS en comparaison avec la reconstruction par génération de colonnes. Comme la méthode générique de LNS à la section 2.4 présente la possibilité d'avoir plusieurs opérateurs de reconstruction, il serait possible d'utiliser les deux méthodes de reconstruction en alternance et possiblement d'autres méthodes de reconstruction afin d'ajouter de la diversité à la phase de reconstruction.

Il existe, de plus, de nombreuses variantes de problèmes de tournées de véhicules qui n'ont pas été abordées dans cette thèse. Le problème de livraison d'huile de chauffage présenté au chapitre 6, par exemple, concerne la livraison d'un seul produit alors qu'en pratique, les distributeurs ont souvent à livrer plusieurs produits dans des camions à plusieurs compartiments, ce qui complique grandement la résolution du problème.

Finalement, il serait pertinent d'appliquer la méthode à d'autres contextes que des problèmes de tournées de véhicules. Tel que mentionné à la section 2.5, il existe plusieurs autres types de problèmes pouvant être résolu par une méthode de génération de colonnes. Certains aspects de la méthode étaient spécifiques aux problèmes de tournées de véhicules, comme la résolution de sous-problèmes par la méthode tabou ou les opérateurs de destruction. De

nouvelles approches pour ces aspects doivent être développées pour de nouveaux problèmes, mais la méthode hybride générale reste la même.

RÉFÉRENCES

ADELMAN, D. (2004). A price-directive approach to stochastic/inventory routing. *Operations Research*, vol. 52, pp. 499–514.

ANGELELLI, E. et SPERANZA, M. (2002). The periodic vehicle routing problem with intermediate facilities. *European Journal of Operational Research*, vol. 137, pp. 233–247.

APPLEGATE, D. et COOK, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, vol. 3, pp. 149–156.

ARCHETTI, C., BERTAZZI, A., HERTZ, A. et SPERANZA, M. (2009a). A hybrid heuristic for an inventory-routing problem. Technical Report n. 317, Department of Quantitative Methods, University of Brescia, Italie.

ARCHETTI, C., BIANCHESSI, N. et SPERANZA, M. (2009b). A Column Generation Approach for the Split Delivery Vehicle Routing Problem. *À paraitre dans Networks*.

ARCHETTI, C. et SAVELSBERGH, M. (2009). The trip scheduling problem. *Transportation Science*, vol. 43, pp. 417–431.

BALDACCI, R., BARTOLINI, E., MINGOZZI, A. et ROBERTI, R. (2010). An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*.

BARD, J., KONTORAVDIS, G. et YU, G. (2002). A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, vol. 36, pp. 250–269.

BARNHART, C., BOLAND, N., CLARKE, L., JOHNSON, E., NEHMAUSER, G. et SHENOI, R. (1998a). Flight string model for aircraft fleeting and routing. *Transportation Science*, vol. 32, pp. 208–220.

BARNHART, C., JOHNSON, E., NEMHAUSER, G., SAVELSBERGH, M. et VANCE, P. (1998b). Branch-and-price : column generation for solving huge integer programs. *Operations Research*, vol. 46, pp. 316–329.

BARTODZIEJ, P., DERIGS, U., MALCHEREK, D. et VOGEL, U. (2009). Models and algorithms for solving combined vehicle and crew scheduling with rest contraints : An application to road feeder service planning in air cargo transportation. *OR Spectrum*, vol. 31, pp. 405–429.

BENT, R. et VAN HENTENRYCK, P. (2004). A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, vol. 38, pp. 515–530.

BERGER, J., BARKAOUI, M. et BRÄYSY, O. (2003). A route-directed hybrid genetic approach for the vehicle routing problem with time windows. *INFOR*, vol. 41, pp. 179–194.

BERTAZZI, L., SAVELSBERGH, M. et SPERANZA, M. (2008). Inventory Routing. S. R. B. Golden et E. Wasil, éditeurs, *The Vehicle Routing Problem : Latest Advances and New Challenges*, Springer, New York. pp. 49–72.

BOLAND, N., DETHRIDGE, J. et DUMITRESCU, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, vol. 34, pp. 58–68.

BOSCHETTI, M., MANIEZZO, V. et ROFFILLI, M. (2009). Decomposition techniques as metaheuristic frameworks. V. Maniezzo, T. Stützle et S. Voß, éditeurs, *Matheuristics : Hybridizing Metaheuristics and Mathematical Programming*, Springer, Heidelberg, vol. 10 de *Annals of Information Systems*. pp. 135–158.

BOSTEL, N., DEJAX, P., GUEZ, P. et TRICOIRE, F. (2008). Multiperiod planning and routing on a rolling horizon for field force optimization logistics. B. Golden, S. Raghavan et E. Wasil, éditeurs, *The Vehicle Routing Problem : Latest Advances and New Challenges*, Springer, New York. pp. 503–525.

BOUBAKER, K., DESAULNIERS, G. et ELHALLAOUI, I. (2010). Bidline scheduling with equity by heuristic dynamic constraint aggregation. *Transportation Research Part B*, vol. 44, pp. 50–61.

BRÄYSY, O. (2003). A reactive variable neighborhood search for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, vol. 15, pp. 347–368.

BRÄYSY, O. et GENDREAU, M. (2005a). Vehicle routing problem with time windows, Part I : Route construction and local search algorithms. *Transportation Science*, vol. 39, pp. 104–118.

BRÄYSY, O. et GENDREAU, M. (2005b). Vehicle routing problem with time windows, Part II : Metaheuristics. *Transportation Science*, vol. 39, pp. 119–139.

CARCHRAE, T. et BECK, J. (2005). Cost-based large neighborhood search. *Workshop on the Combination of Metaheuristic and Local Search with Constraint Programming Techniques (MLS+CP)*, Université de Nantes, France.

CASEAU, Y. et LABURTHE, F. (1999). Effective forget-and-extend heuristics for scheduling problems. *Proceedings of CP-AI-OR'1999*. Ferrara, Italie.

CHABRIER, A. (2006). Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, vol. 33, pp. 2972–2990.

CHIANG, W. et RUSSELL, R. (1997). A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, vol. 9, pp. 417–430.

CHRISTIANSEN, C. et LYSGAARD, J. (2007). A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research Letters*, vol. 35, pp. 773–781.

CHRISTIANSEN, M. et NYGREEN, B. (2006). A method for solving ship routing problems with inventory constraints. *Annals of Operations Research*, vol. 81, pp. 357–378.

CLARKE, G. et WRIGHT, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, vol. 12, pp. 568–581.

COOK, W. et RICH, J. (1999). A parallel cutting-plane algorithm for the vehicle routing problem with time windows. Rapport technique TR99-04, Department of Computational and Applied Mathematics, Rice University, Houston.

CORDEAU, J.-F., DESAULNIERS, G., LINGAYA, N., SOUMIS, F. et DESROSIERS, J. (2001a). Simultaneous locomotive and car assignment at VIA Rail Canada. *Transportation Research Part B*, vol. 35, pp. 767–787.

CORDEAU, J.-F., GENDREAU, M. et LAPORTE, G. (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, vol. 30, pp. 105–119.

CORDEAU, J.-F., LAPORTE, G. et MERCIER, A. (2001b). A unified tabu search heuristic for the vehicle routing problems with time windows. *Journal of the Operational Research Society*, vol. 52, pp. 928–936.

CORDEAU, J.-F., LAPORTE, G., SAVELSBERGH, M. et VIGO, D. (2007). Vehicle Routing. C. Barnhart et G. Laporte, éditeurs, *Transportation*, North-Holland, Amsterdam, vol. 14 de *Handbooks in Operations Research and Management Science*. pp. 410–417.

CREVIER, B., CORDEAU, J.-F. et LAPORTE, G. (2007). The multi-depot vehicle routing problem with inter-depot routes. *European Journal of Operational Research*, vol. 176, pp. 756–773.

DANNA, E. et LE PAPE, C. (2005). Branch-and-price heuristics : A case study on the vehicle routing problem with time windows. G. Desaulniers, J. Desrosiers et M. Solomon, éditeurs, *Column Generation*, Springer, New York. pp. 99–129.

DANTZIG, G. et RAMSER, J. (1959). The truck dispatching problem. *Management Science*, vol. 6, pp. 80–91.

DANTZIG, G. et WOLFE, P. (1960). Decomposition principle for linear programs. *Operations Research*, vol. 8, pp. 101–111.

DE FRANCESCHI, R., FISCHETTI, M. et TOTH, P. (2006). A new ILP-based refinement heuristic for vehicle routing problems. *Mathematical Programming B*, vol. 105, pp. 471–499.

DESAULNIERS, G. (2010). Branch-and-price-and-cut for the split delivery vehicle routing problem with time windows. *Operations Research*, vol. 58, pp. 179–192.

DESAULNIERS, G., DESROSIERS, J., IOACHIM, I., SOUMIS, F., SOLOMON, M. et VILLENEUVE, D. (1998a). A unified framework for time constrained vehicle routing and crew scheduling problems. T. Crainic et G. Laporte, éditeurs, *Fleet Management and Logistics*, Kluwer, Norwell, MA. pp. 57–93.

DESAULNIERS, G., DESROSIERS, J., IOACHIM, I., SOUMIS, F., SOLOMON, M. et VILLENEUVE, D. (2002). Accelerating strategies for column generation methods in vehicle routing and crew scheduling problems. C. Ribeiro et P. Hansen, éditeurs, *Essays and Surveys in Metaheuristics*, Kluwer, Norwell, MA. pp. 390–324.

DESAULNIERS, G., DESROSIERS, J. et SOLOMON, M., éditeurs (2005). *Column generation*. Springer.

DESAULNIERS, G., DESROSIERS, J., SOLOMON, M. et SOUMIS, F. (1997). Daily aircraft routing and scheduling. *Management Science*, vol. 43, pp. 841–855.

DESAULNIERS, G., LAVIGNE, J. et SOUMIS, F. (1998b). Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research*, vol. 111, pp. 479–494.

DESAULNIERS, G., LESSARD, F. et HADJAR, A. (2008). Tabu search, generalized k-path inequalities, and partial elementarity for the vehicle routing problem with time windows. *Transportation Science*, vol. 42, pp. 387–404.

DESROCHERS, M., DESROSIERS, J. et SOLOMON, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, vol. 40, pp. 342–534.

DESROCHERS, M. et SOUMIS, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, vol. 23, pp. 1–13.

DESROSIERS, J., DUMAS, Y., SOLOMON, M. et SOUMIS, F. (1995). Time constrained routing and scheduling. M. Ball, T. Magnanti, C. Monma et G. Nemhauser, éditeurs, *Network Routing*, Elsevier Science, Amsterdam, vol. 8 de *Handbooks in Operations Research and Management Science*. pp. 35–139.

DESROSIERS, J. et LÜBBECKE, M. (2005). A primer in column generation. G. Desaulniers, J. Desrosiers et M. Solomon, éditeurs, *Column Generation*, Springer, New York. pp. 1–32.

DREXL, M. et PRESCOTT-GAGNON, E. (2010). Labelling algorithms for the elementary shortest path problem with resource constraints considering EU drivers' rules. *Logistics Research*, vol. 2, pp. 79–96.

DROR, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, vol. 42, pp. 977–978.

DROR, M. (2005). Routing propane deliveries. A. Langevin et D. Riopel, éditeurs, *Logistics Systems : Design and Optimization*, Springer, New York. pp. 299–322.

DROR, M. et BALL, M. (1987). Inventory/routing : Reduction from annual to a short period problem. *Naval Research Logistics*, vol. 34, pp. 891–905.

DROR, M., BALL, M. et GOLDEN, B. (1985). Computational comparison of algorithms for inventory routing. *Annals of Operations Research*, vol. 4, pp. 3–23.

DUMAS, Y., DESROSIERS, J. et SOUMIS, F. (1991). The pickup and delivery problem with time windows. *European Journal of Operations Research*, vol. 54, pp. 7–22.

EUROPEAN UNION (2002). Directive 2002/15/EC of the European Parliament and of the Council of 11 March 2002 on the organisation of the working time of persons performing mobile road transport activities. *Official Journal of the European Union*. L 080, 23.03.2002.

EUROPEAN UNION (2006). Regulation (EC) No 561/2006 of the European Parliament and of the Council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport and amending Council Regulations (EEC) No 3821/85 and (EC) No 2135/98 and repealing Council Regulation (EEC) No 3820/85. *Official Journal of the European Union*. L 102, 11.04.2006.

FEILLET, D., DEJAX, P., GENDREAU, M. et GUEGUEN, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints : Application to some vehicle routing problems. *Networks*, vol. 44, pp. 216–229.

FEILLET, D., GENDREAU, M. et ROUSSEAU, L.-M. (2007). New refinements for the solution of vehicle routing problems with branch and price. *Information System and Operations Research (INFOR)*, vol. 45, pp. 239–256.

GAMACHE, M., SOUMIS, F., MARQUIS, G. et DESROSIERS, J. (1999). A column generation approach for large scale aircrew rostering problems. *Operations Research*, vol. 47, pp. 247–263.

GEHRING, H. et HOMBERGER, J. (1999). A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. K. Miettinen, M. Makela et J. Toivanen, éditeurs, *Proceeding of EUROGEN99 - Short Course on Evolutionary Algorithms in Engineering and Computer Science*, Université de Jyväskylä, Jyväskylä, Finlande. pp. 57–64.

GEHRING, H. et HOMBERGER, J. (2001). A parallel two-phase metaheuristic for routing problems with time windows. *Asia-Pacific Journal of Operational Research*, vol. 18, pp. 35–47.

GENDREAU, M., DEJAX, P., FEILLET, D. et GUEGUEN, C. (2006). Vehicle routing with time windows and split deliveries. Rapport Technique 2006-851, Laboratoire Informatique d'Avignon, France.

GENDREAU, M., HERTZ, A. et LAPORTE, G. (1992). A new insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, vol. 40, pp. 1086–1093.

GILMORE, P. et GOMORY, R. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, vol. 9, pp. 849–859.

GLOVER, F. (1991). Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem. Working paper, College of Business and Administration. Université du Colorado, Boulder, CO.

GLOVER, F. (1992). New ejection chain and alternating path methods for traveling salesman problems. O. Balci, R. Sharda et S.Zenios, éditeurs, *Computer Science and Operations Research : New Developments in their Interfaces*, Pergamon Press, Oxford, Royaume-Uni. pp. 449–509.

GLOVER, F. et LAGUNA, M. (1997). *Tabu search*. Kluwer, Norwell, MA.

GODARD, D., LABORIE, P. et NUIJTEN, W. (2005). Randomized large neighborhood search for cumulative scheduling. S. B. K. Myers et K. Rajan, éditeurs, *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS)*. Monterey, California, USA, pp. 81–89.

GOEL, A. (2009). Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, vol. 43, pp. 17–26.

GOEL, A. (2010). Truck driver scheduling in the European Union. *Transportation Science*, vol. 44, pp. 429–441.

GOEL, A. et GRUHN, V. (2005). Large neighborhood search for the rich VRP with multiple pickup and delivery locations. *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*. Puerto de la Cruz, Tenerife, Espagne.

GOEL, A. et GRUHN, V. (2006). Drivers' working hours in vehicle routing and scheduling. *Proceedings of the 9th IEEE International Conference on Intelligent Transportation Systems (ITSC 2006)*. Toronto, Canada, pp. 1280–1285.

GOEL, A. et KOK, L. (2009). Efficient truck driver scheduling in the United States. Document de travail, University of Twente, Twente, Pays-Bas.

GOLDEN, B., RAGHAVAN, S. et WASIL, E., éditeurs (2008). *The vehicle routing problem : Latest advances and new challenges*. Operations Research/Computer Science Interfaces Series. Springer, New York, NY.

GRONHAUG, R., CHRISTIANSEN, M., DESAULNIERS, G. et DESROSIERS, J. (2010). A branch-and-price method for a liquefied natural gas inventory routing problem. *Transportation Science*, vol. 44, pp. 400–415.

GUTIÉRREZ-JARPA, G., DESAULNIERS, G., LAPORTE, G. et MARIANOV, V. (2010). A branch-and-price algorithm for the vehicle routing problem with deliveries, selective pickups and time windows. *European Journal of Operational Research*, vol. 206, pp. 341–349.

HAASE, K., DESAULNIERS, G. et DESROSIERS, J. (2001). Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, vol. 35, pp. 286–303.

HASHIMOTO, H. et YAGIURA, M. (2008). A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows. *EvoCOP'08 : Proceedings of the 8th European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer-Verlag, Berlin, Heidelberg, pp. 254–265.

HOMBERGER, J. et GEHRING, H. (2005). A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, vol. 162, pp. 220–238.

HUISMAN, D. (2007). A column generation approach to solve the rail crew re-scheduling problem. *European Journal of Operational Research*, vol. 180, pp. 163–173.

IBARAKI, T., IMAHORI, S., KUBO, M., MASUDA, T., UNO, T. et YAGIURA, M. (2005). Effective local search algorithms for routing and scheduling problems with general time window constraints. *Transportation Science*, vol. 39, pp. 206–232.

IBARAKI, T., IMAHORI, S., NONOBE, K., SOBUE, K., UNO, T. et YAGIURA, M. (2008). An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Applied Mathematics*, vol. 156, pp. 2050–2069.

IOANNOU, G., KRITIKIS, M. et PRASTACOS, G. (2001). A greedy look-ahead heuristic for the vehicle routing problem with time windows. *Journal of the Operational Research Society*, vol. 52, pp. 523–537.

IRNICH, S. (2008). Resource extension functions : Properties, inversion, and generalization to segments. *OR Spectrum*, vol. 30, pp. 113–148.

IRNICH, S. et DESAULNIERS, G. (2005). Shortest path problems with resource constraints. G. Desaulniers, J. Desrosiers et M. Solomon, éditeurs, *Column Generation*, Springer, New York, NY. pp. 33–65.

IRNICH, S. et VILLENEUVE, D. (2006). The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, vol. 18, pp. 391–406.

JEPSEN, M., PETERSEN, B., SPOORENDONK, S. et PISINGER, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, vol. 56, pp. 497–511.

KALLEHAUGE, B., BOLAND, N. et MADSEN, O. (2007). Path inequalities for the vehicle routing problem with time windows. *Networks*, vol. 49, pp. 273–293.

KALLEHAUGE, B., LARSEN, J., MADSEN, O. et SOLOMON, M. (2005). Vehicle routing problem with time windows. G. Desaulniers, J. Desrosiers et M. Solomon, éditeurs, *Column Generation*, Springer, New York, NY. pp. 67–98.

KLABJAN, D., JOHNSON, E., NEMHAUSER, G., GELMAN, E. et RAMASWAMY, S. (2002). Airline crew scheduling with time windows and plane count constraints. *Transportation Science*, vol. 36, pp. 337–348.

KLEYWEGT, A., NORI, V. et SAVELSBERGH, M. (2002). The stochastic inventory routing problem with direct deliveries. *Transportation Science*, vol. 36, pp. 94–118.

KOHL, N., DESROSIERS, J., MADSEN, O., SOLOMON, M. et SOUMIS, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, vol. 33, pp. 101–116.

KOK, A., MEYER, C., KOPFER, H. et SCHUTTEN, J. (2010). A Dynamic Programming Heuristic for the Vehicle Routing Problem with Time Windows and European Community Social Legislation. *Transportation Science*, vol. 44, pp. 442–454.

LABORIE, P. et GODARD, D. (2007). Self-adapting large neighborhood search : Application to single-mode scheduling problems. *Proceedings of MISTA-07*. Paris, France, pp. 276–284.

LÜBBECKE, M. et DESROSIERS, J. (2005). Selected topics in column generation. *Operations Research*, vol. 53, pp. 1007–1023.

LE BOUTHILLIER, A., CRAINIC, T. et KROPF, P. (2005). A guided cooperative search for the vehicle routing problem with time windows. *IEEE Intelligent Systems*, vol. 20, pp. 36–42.

LIM, A. et ZHANG, X. (2007). A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, vol. 19, pp. 443–457.

LYSGAARD, J. (2006). Reachability cuts for the vehicle routing problem with time windows. *European Journal of Operational Research*, vol. 175, pp. 210–223.

MANIEZZO, V., STÜTZLE, T. et VOß, S., éditeurs (2009). *Matheuristics : Hybridizing Metaheuristics and Mathematical Programming*, vol. 10 de *Annals of Information Systems*. Springer, Heidelberg.

MESTER, D. et BRÄYSY, O. (2005). Active guided evolution strategies for large scale vehicle routing problem with time windows. *Computers & Operations Research*, vol. 32, pp. 1592–1614.

MOIN, N. et SALHI, S. (2007). Inventory routing problem : A logistical overview. *Journal of the Operational Research Society*, vol. 58, pp. 1185–1194.

MOURGAYA, M. et VANDERBECK, F. (2007). Column generation based heuristic for tactical planning in multi-period vehicle routing. *European Journal of Operational Research*, vol. 183, pp. 1028–1041.

NAGATA, Y., BRÄYSY, O. et DULLAERT, W. (2010). A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, vol. 37, pp. 724–737.

PEETERS, M. et KROON, L. (2008). Circulation of railway rolling stock : a branch-and-price approach. *Computers & Operations Research*, vol. 35, pp. 538–556.

PEPIN, A.-S., DESAULNIERS, G., HERTZ, A. et HUISMAN, D. (2009). A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of Scheduling*, 12, pp. 17–30.

PERRON, L., SHAW, P. et FURNON, V. (2004). Propagation guided large neighborhood search. *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming*. Toronto, Canada, pp. 468–481.

PISINGER, D. et ROPKE, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, vol. 34, pp. 2403–2435.

POTVIN, J.-Y. et ROUSSEAU, J.-M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, vol. 66, pp. 331–340.

PRESCOTT-GAGNON, E., DESAULNIERS, G., DREXL, M. et ROUSSEAU, L.-M. (2010). European driver rules in vehicle routing with time windows. *Transportation Science*, vol. 44, pp. 455–473.

PRESCOTT-GAGNON, E., DESAULNIERS, G. et ROUSSEAU, L.-M. (2009). A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, vol. 54, pp. 190–204.

PUCHINGER, J. et RAIDL, G. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization : A survey and classification. *Artificial Intelligence and Knowledge Engineering Applications : A Bioinspired Approach*, Springer, New York, NY. pp. 41–53.

REPOUSSIS, P., TARANTILIS, C. et IOANNOU, G. (2009). Arc-guided evolutionary algorithm for the vehicle routing problem with time windows. *IEEE Transactions on Evolutionary Computation*, vol. 13, pp. 624–647.

RIBEIRO, C. et SOUMIS, F. (1994). Column generation approach to the multiple depot vehicle scheduling problem. *Operations Research*, vol. 42, pp. 41–52.

RIGHINI, G. et SALANI, M. (2006). Symmetry helps : Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, vol. 3, pp. 255–273.

RIGHINI, G. et SALANI, M. (2008). New dynamic programming algorithms for the resource-constrained elementary shortest path problem. *Networks*, vol. 51, pp. 155–170.

ROPKE, S. et CORDEAU, J.-F. (2009). Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, vol. 43, pp. 267–286.

ROUSSEAU, L.-M., GENDREAU, M. et PESANT, G. (2002). Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of Heuristics*, vol. 8, pp. 43–58.

SAVELSBERGH, M. (1992). The vehicle routing problem with time windows : Minimizing route duration. *ORSA Journal on Computing*, vol. 4, pp. 146–154.

SAVELSBERGH, M. et SOL, M. (1995). The general pickup and delivery problem. *Transportation Science*, vol. 29, pp. 17–29.

SAVELSBERGH, M. et SOL, M. (1998). Drive : Dynamic routing of independent vehicles. *Operations Research*, vol. 46, pp. 474–490.

SCHRIMPF, G., SCHNEIDER, J., STAMM-WILMBRANDT, H. et DUECK, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, vol. 159, pp. 139–171.

SHAW, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. M. Maher et J. Puget, éditeurs, *Proceedings of Constraints Programming '98*. Springer-Verlag, Berlin, pp. 417–431.

SOLOMON, M. (1987). Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, vol. 35, pp. 254–265.

TAILLARD, E., BADEAU, P., GENDREAU, M. et POTVIN, J. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, vol. 31, pp. 170–186.

TARANTILIS, C., ZACHARIADIS, E. et KIRANOUDIS, C. (2008). A hybrid guided local search for the vehicle routing problem with intermediate replenishment facilities. *INFORMS Journal on Computing*, vol. 20, pp. 154–168.

TOTH, P. et VIGO, D., éditeurs (2002). *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia.

VANCE, P., BARNHART, C., JOHNSON, E. et NEMHAUSER, G. (1997). Airline crew scheduling : A new formulation and decomposition algorithm. *Operations Research*, vol. 45, pp. 188–200.

XU, H., CHEN, Z., RAJAGOPAL, S. et ARUNAPURAM, S. (2003). Solving a practical pick-up and delivery problem. *Transportation Science*, vol. 37, pp. 347–364.

ZIARATI, K., SOUMIS, F., DESROSIERS, J., GÉLINAS, S. et SAINTONGE, A. (1997). Locomotive Assignment with Heterogeneous Consists at CN North America. *European Journal of Operational Research*, vol. 97, pp. 281–292.

ZÄPFEL, G. et BÖGL, M. (2008). Multi-period vehicle routing and crew scheduling with outsourcing options. *International Journal of Production Economics*, vol. 113, pp. 980–996.